Institutional Repository - Research Portal
Dépôt Institutionnel - Portail de la Recherche
researchportal.unamur.be
University of Namur

THESIS / THÈSE

MASTER EN SCIENCES INFORMATIQUES

Rétro-ingénierie de bases de données relationnelles et CODASYL

Bellem, Jean-François; Deflorenne, Xavier

Award date:
1993

Awarding institution:
Universite de Namur

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 05. May. 2023

Facultés Universitaires Notre-Dame de la Paix
Institut d'Informatique
Rue Grandgagnage, 21b
B-5000 NAMUR

Rétro-ingénierie de bases de données relationnelles et CODASYL

Jean-François BELLEM
&
Xavier DEFLORENNE

Promoteur : Jean-Luc Hainaut

Mémoire présenté en vue
de l'obtention du grade de
Licencié et Maître en Informatique

Année académique 1992 - 1993

ABSTRACT

En quelques mots, la rétro-ingénierie est un nouveau domaine de la conception des bases de données. Elle est une démarche semi-automatique, utilisée surtout dans la re-documentation et la modification des systèmes d'informations. Le présent ouvrage se propose de jeter les bases d'une méthodologie de la rétro-ingénierie et de les appliquer à deux systèmes de gestion de bases de données; SQL et CODASYL. Cette réflexion est illustrée par de nombreux exemples tirés de la littérature et de cas réels. Certaines phases du processus de rétro-ingénierie ont été implémentées dans un outil CARE développé par une équipe de recherche de l'université de Namur.

In a few words, reverse engineering is a brand new domain of the databases conception. It's a half-automatic approach used in re-documentation or modification of informations systems. This works intends to establish the bases of a methodology for the reverse-engineering applied to two databases management systems : SQL and CODASYL. This thinking is illustrated with plenty of examples taken from the literature and real cases. Some phases of the reverse engineering process have been implemented in a CARE tool developed by a team of searchers in the university of Namur.

REMERCIEMENTS

Nous tenons à exprimer notre gratitude envers Monsieur le professeur Jean-Luc Hainaut pour ses conseils, ses critiques et son soutien.

Nous sommes également reconnaissant envers la société BIM de Bruxelles pour nous avoir accueilli durant notre stage et tout particulièrement envers Madame Isabelle de Zeger ainsi que Messieurs Jean-Louis Binot et Roger Fisset.

Nous tenons de même à remercier Monsieur Olivier Masson au nom de la société Glaverbel, Monsieur Stéphane Ries de la banque BUCL du Luxembourg et le centre de calcul des FUNDP pour leur coopération.

Merci aussi au groupe de travail Tramis; Messieurs Olivier Marchand, Jean-Marc Hick et Jean Henrard.

Merci enfin à nos familles respectives pour leur soutien moral et logistique.

TABLE DES MATIÈRES

1. INTRODUCTION

La finalité de ce mémoire est de servir de base à toutes personnes désirant procéder à la rétro-ingénierie d'une base de données relationnelle ou CODASYL.

Pour ce faire, après avoir développé l'état de l'art de la méthodologie (chapitre 2), nous nous attacherons aux bases de données relationnelles (chapitre 3) et CODASYL (chapitre 4). Pour chacune de celles-ci, nous ferons un bref rappel de leurs caractéristiques. Ensuite, nous identifierons les sources d'information qu'elles peuvent fournir. Enfin, nous appliquerons aux cas particuliers que sont ces deux types de base de données, la partie de la méthodologie dépendante d'un modèle de représentation des données.

Le cinquième chapitre, pour sa part, sera dédié à la partie de la méthodologie qui est indépendante du type de base de données traitée.

Enfin, le sixième et dernier chapitre recueillera la synthèse de notre apport à l'atelier logiciel TRAMIS.

1.1. DÉFINITION ET OBJECTIFS DE LA RÉTRO-INGÉNIERIE

1.1.1. DÉFINITION

La rétro-ingénierie est un sous-domaine du software engineering. Il a pour but de retrouver la structure logique et fonctionnelle d'une application en se basant sur l'analyse de celle-ci.

Un sous-ensemble de la rétro-ingénierie se focalise sur l'étude des données utilisées par l'application.

La rétro-ingénierie n'étant encore qu'à l'aube de son développement, nous avons choisi de porter notre intérêt sur ce sous-ensemble. Ce choix est dû au fait que, dans le temps, la structure des données utilisées par l'application est plus stable que les programmes eux-mêmes. En effet, ces données représentent les concepts utilisés dans le domaine de l'application, tandis que les programmes ne représentent qu'une façon de les traiter. De ce fait, pour comprendre une application, il est d'abord primordial de comprendre les données qu'elle manipule.

Nous définirons donc la rétro-ingénierie comme un processus qui, sur base de la définition d'une base de données en termes de concepts de bas niveau (niveau physique et partiellement logique), tente de l'exprimer à un niveau d'abstraction supérieur (logique ou conceptuel selon l'objectif que l'on a en utilisant ce processus).

En d'autres termes, un processus de rétro-ingénierie tente d'extraire l'abstrait du concret dans le but d'une meilleure compréhension. Il est à noter que le modèle des données dans lequel on veut représenter l'abstrait devra être au minimum aussi riche que le modèle sur lequel repose l'expression du concret.

Après avoir défini ce qu'est un processus de rétro-ingénierie, nous voudrions spécifier ce qu'il n'est pas :

- Un processus capable de corriger des fautes de conception.

- Un processus entièrement automatisable. Car le processus de conception étant semi-formel (et donc semi-créatif), nous rencontrerions de grandes difficultés à vouloir automatiser son inverse.

- Une procédure retrouvant LE schéma conceptuel utilisé lors de la phase de conception. En effet, le schéma conceptuel que nous tenterons de retrouver sera un des schémas possibles de l'expression de la base de données. Ceci est dû au fait que plusieurs constructions conceptuelles peuvent être traduites en une même expression concrète.

1.1.2. LES OBJECTIFS

1.1.2.1. REMÉDIER À UN MANQUE DE DOCUMENTATION

Une documentation claire, pertinente et correcte est un outil indispensable pour l'utilisation d'une base de données. En effet, elle permettra de prendre connaissance des règles régissant cette dernière et cela tout au long de son cycle de vie.

Mais il se peut que cette documentation soit inexistante. L'utilisation de méthodologies pour le développement d'applications n'a pas toujours fait l'unanimité dans les services informatiques de nombreuses entreprises. Dans ce cas, un processus de rétro-ingénierie semble bien approprié pour résorber cette carence en permettant de redocumenter l'application notamment via un schéma logique et/ou un schéma conceptuel réalisé à partir de la définition de la base de données.

1.1.2.2. AMÉLIORER L'UTILISATION DE L'APPLICATION

Nombreuses sont les applications informatiques qui se sont développées petit à petit, version après version, sans que la documentation ne suive l'évolution de l'application. De ce fait, on perd vite la maîtrise de tels "monstres".

La seule solution pour pouvoir continuer à évoluer est de restructurer l'application. Cela afin d'obtenir une meilleure version en terme de documentation, structure, performance, maintenance, ...

Pour ce faire, il faudra passer par la rétro-ingénierie de l'application.

1.1.2.3. ACCOMPAGNER L'ÉVOLUTION DE L'APPLICATION

Toute application est appelée à évoluer. Que ce soit pour une modification ou une extension, le travail de maintenance se fait tout au long de la vie de l'application.

La modification peut porter sur une partie des traitements ou être le changement du SGBD sur lequel l'application repose.

Dans le premier cas, les modifications de certains traitements pour cause d'erreurs ou de légères modifications dans le domaine d'application, peuvent nécessiter une redocumentation.

Dans le second cas, il faudra retrouver le schéma logique (si la nouvelle base de données est de la même famille que l'ancienne) ou conceptuel (dans le cas contraire) pour, ensuite, implémenter à nouveau la base de données. En effet, le passage d'une base de données à l'autre ne peut se faire sans repasser par un niveau d'abstraction supérieur. Sinon, on risque de retrouver des constructions d'optimisation dédiées à la base de données initiale dans le schéma de la nouvelle base de données.

L'extension de l'application, quant à elle, se fera soit par ajouts de modules d'où souvent une extension de la base de données soit par l'intégration de différentes applications ce qui nécessite forcément une intégration des différentes bases de données qu'elles utilisent. Cette extension devra être gérée aux niveaux conceptuel et logique d'où l'intérêt de posséder une documentation à jour.

1.2. MÉTHODOLOGIE DE CONCEPTION DES BASES DE DONNEES

Pour bien élaborer un processus de rétro-ingénierie, il faut d'abord avoir une bonne compréhension du processus de conception d'applications informatiques. C'est pour cela que nous nous permettons de rappeler les points essentiels de ce dernier. Mais avant d'entamer ce rappel, nous soulignons le fait qu'un processus de conception n'est pas entièrement formel, la créativité du concepteur y prenant une place prépondérante. Cette semi formalité est une des principales difficultés dans l'élaboration d'un processus de rétro-ingénierie car elle ne permet que des hypothèses et empêche toute automatisation complète.

La méthode que nous nous proposons de décrire est inspirée de **[Bat92]** et **[Hai93]**. Elle se découpe en trois niveaux : conceptuel, logique et physique. Cette découpe en niveaux vient de l'objectif de réduire la complexité du processus. Chaque niveau ayant un type de décision différent ainsi que des degrés d'abstraction décroissants, allant d'un haut niveau (abstrait) à un niveau de description technique (concret). De plus, chaque niveau reposera sur un type de schéma qui lui est propre et qui est construit en accord avec un modèle de données adéquat (concepts, opérateurs et règles sur les concepts).

Remarquons que cette approche est orientée vers les données, c'est à dire que l'on conçoit d'abord la base de données et puis les applications qui l'utilisent, ce qui convient très bien à la définition que l'on s'est donné de la rétro-ingénierie.

1.2.1. LE NIVEAU CONCEPTUEL

Des trois niveaux, le niveau conceptuel est celui qui a le plus haut degré d'abstraction. Il repose sur l'analyse du domaine de l'application pour produire un schéma conceptuel des données qui devront figurer dans la base de données. Ce haut niveau d'abstraction indique que le but de l'étude conceptuelle est de décrire ces données et non les structures de stockage requises pour les gérer. Donc, nous n'y retrouverons pas de spécification technique ni d'impératif de performance.

Le schéma conceptuel sera décrit par un modèle, nommé modèle conceptuel, de description de données répondant à ses besoins d'abstraction. Le modèle le plus souvent utilisé est le modèle Entité-Association dont les concepts de base sont : Le type d'entité, le type d'association, l'attribut et la contrainte d'intégrité.

Ce processus d'étude conceptuelle se divise en trois sous processus : L'étude des besoins des utilisateurs qui donnera pour chaque utilisateur un premier schéma, la normalisation de ces schémas, puis, après leur validation par les utilisateurs, l'intégration en un seul schéma conceptuel.

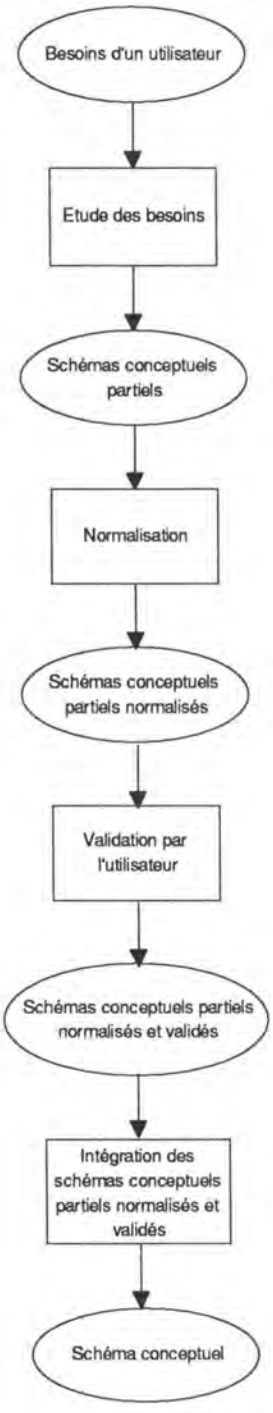

Besoins d'un utilisateur
Etude des besoins
Schémas conceptuels partiels
Normalisation
Schémas conceptuels partiels normalisés
Validation par l'utilisateur
Schémas conceptuels partiels normalisés et validés
Intégration des schémas conceptuels partiels normalisés et validés
Schéma conceptuel

Figure I.1.

1.2.2. LE NIVEAU LOGIQUE

Comme nous le voyons dans la figure I.2. , le processus de conception de niveau logique produit un schéma appelé schéma logique. Ce schéma sera exprimé dans un langage qui constitue le modèle logique et qui est spécifique a une famille de bases de données (relationnelle, réseau, hiérarchique, ...).

Nous remarquerons aussi que, pour traduire le schéma conceptuel, ce processus a besoin d'autres informations :

- Une description du modèle logique de la base de données cible pour que la description représentée par le schéma logique puisse être supportée par cette base de données.

- Des critères de performance requis pour l'implémentation (temps de réponse, volume de la base de données, ...).

- Des informations sur la charge de la base de données c'est à dire sa population et les requêtes effectuées sur celle-ci accompagnées de leur fréquence.

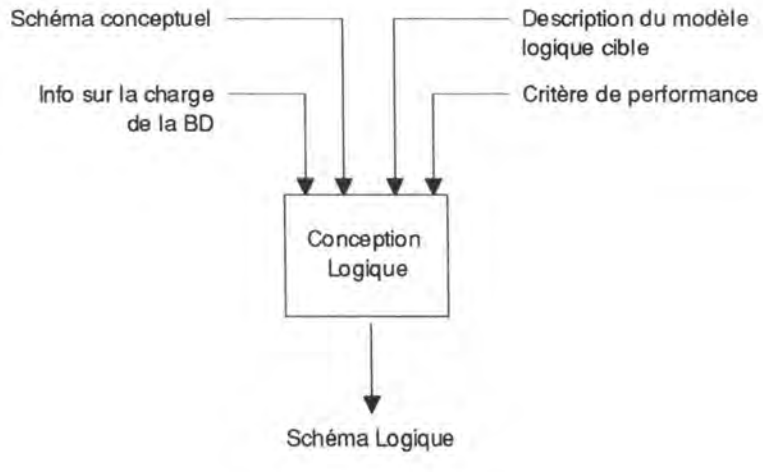


Figure I.2.

Le schéma conceptuel sera donc transformé dans le but de lui donner deux qualités : La compatibilité avec une base de données (ou plutôt avec son modèle) et l'optimisation de son utilisation.

Nous soulignerons donc que contrairement au niveau conceptuel, dont le but fondamental est la complétude et le caractère expressif de son schéma, le niveau logique veut obtenir une représentation qui utilise efficacement le modèle logique tout en préservant la sémantique du schéma conceptuel. Mais comme les modèles logiques n'ont pas un aussi haut niveau d'abstraction que les modèles conceptuels, il faudra utiliser au niveau logique des transformations de mécanismes de représentation de haut niveau propre au modèle conceptuel en terme de structure de plus bas niveau appartenant au modèle logique. Rappelons que ces transformations devront préserver la sémantique. Par transformation préservant la sémantique, nous entendons une transformation dont la construction résultante peut exprimer tout ce qu'exprimait la construction de départ.

Pour atteindre un tel but, on a choisi de diviser le processus de conception de niveau logique en deux sous-processus. Le premier, indépendant du modèle cible, sera chargé de simplifier le schéma conceptuel en remplaçant les constructions de haut niveau d'abstraction. Le second aura comme but de transformer le schéma simplifié en un schéma compatible avec le modèle logique utilisé par le SGBD cible. De plus, ils auront tout deux un rôle d'optimisation.

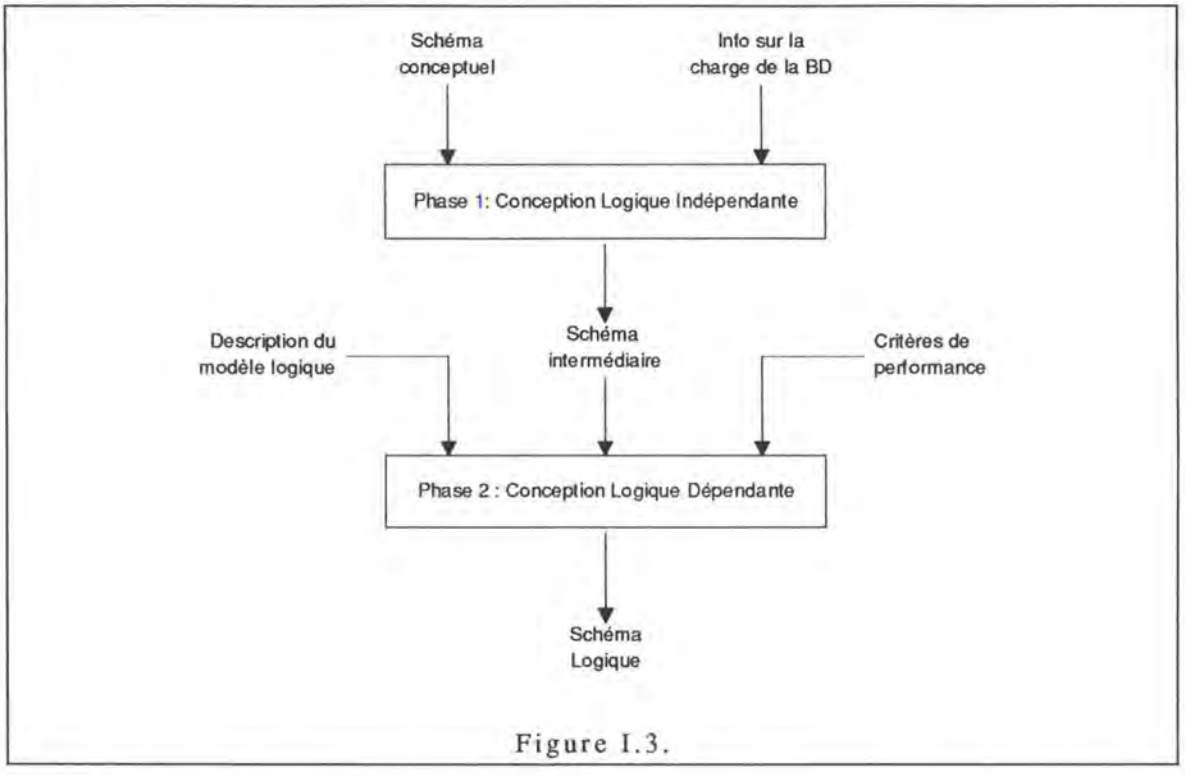


Figure I.3.

1.2.2.1. LE NIVEAU LOGIQUE INDÉPENDANT DU MODELE CIBLE

Le niveau logique indépendant est commun à tous les modèles cibles et a comme résultat un schéma conceptuel simplifié (c'est à dire sans constructions de trop haut niveau d'abstraction tel que les associations plus que binaires, isa, ...) et partiellement optimisé. Cette optimisation est faite sur base de renseignements concernant la charge de la base de données. C'est donc dans ce premier sous-processus qu'aura lieu la prise de décision concernant les données dérivables, la simplification de hiérarchies, la partition ou fusion de types d'entités ou de types d'associations, le choix des clés, ...

1.2.2.2. LE NIVEAU LOGIQUE DÉPENDANT DU MODELE CIBLE

Ce second sous-processus est dirigé par la description du modèle logique cible c'est à dire par les structures de données disponibles dans ce modèle. Il sera donc dépendant du modèle utilisé par la base de données cible mais pas d'une base de données spécifique. Son but principal sera la transformation du schéma intermédiaire en une expression propre au modèle de données décrit en entrée.

1.2.3. LE NIVEAU PHYSIQUE

Ce troisième niveau est le plus proche de la base de données choisie. Il part du schéma logique pour donner en sortie un schéma physique qui n'est autre qu'une description de l'implémentation de la base de données en mémoire secondaire, accompagnée de paramètres de performance.

2. MÉTHODE GÉNÉRALE

2.1. MÉTHODOLOGIE DE LA RÉTRO-INGÉNIERIE

2.1.1. UN MODÈLE UNIQUE DE REPRÉSENTATION DES DONNÉES

Ce chapitre est inspiré de **[Hai93b]**.

Comme nous l'avons vu dans le chapitre 1.1., nous allons devoir, au cours d'un processus de rétro-ingénierie, manipuler (construire, transformer, traduire, ...) des schémas reposant sur des modèles de représentation des données différents. Ces modèles se différencient essentiellement sur leur niveau d'abstraction c'est à dire sur leurs possibilités de représenter des structures de données abstraites.

De ce fait et vu l'approche transformationnelle choisie, il semble utile de baser notre démarche sur un seul et unique modèle de représentation des données qui couvrirait nos besoins de spécification de structures de données. De plus, comme nous le verrons plus loin, notre volonté d'élaborer certains raisonnements indépendamment d'un modèle d'implémentation précis corrobore notre choix d'un modèle unique.

Comme tout modèle, celui que nous choisirons sera composé de concepts, de règles sur ces concepts et d'opérateurs servant à manipuler ces concepts. Mais ce modèle et ces opérateurs seront l'objet de différentes attentes.

Premièrement, ils devront être aptes à supporter aussi bien un processus de rétro-ingénierie qu'un processus de forward engineering. Et ce, pour qu'il soit possible d'intégrer en un seul atelier logiciel un outil CASE et un outil CARE.

Deuxièmement, ils devront exprimer et manipuler des schémas conceptuels, logiques et physiques (les niveaux d'abstraction traditionnels). Ce modèle unique devra donc comprendre aussi bien des concepts abstraits que techniques. Selon le niveau d'abstraction du schéma que l'on voudra représenter par ce modèle, certains concepts seront ignorés alors que d'autres seront utilisés.

Enfin, ils devront inclure les concepts et opérateurs des différents modèles proposés par les bases de données commercialisées. C'est a dire les modèles hiérarchique, réseau et relationnel ainsi que le modèle cobol.

Notre choix s'est porté sur un modèle entité-association étendu. Ce modèle inclura le modèle entité-association classique plus des concepts techniques tels que :

- fichier
- clé d'accès
- chemin d'accès
- clé étrangère
- ...

Nous souhaitons faire remarquer que ce modèle doit être considéré comme un modèle générique capable d'être spécialisé en un modèle par une non utilisation d'un sous-ensemble de ses concepts.

2.1.2. Les étapes d'une démarche de rétro-ingénierie

Ce chapitre est inspiré de **[Hai93b]**.

Le moment est venu de présenter les "guidelines" de la démarche que nous proposons. Avant d'entrer dans les détails, nous allons d'abord analyser les informations en entrée et les résultats d'un processus de rétro-ingénierie.

En ce qui concerne les informations en entrée, une des premières questions que nous nous sommes posés en abordant le sujet était : "Quand je suis face à une application non documentée, où puis je trouver de l'information sur les données qu'elle traite?" A notre grand étonnement, ces sources d'information n'étaient pas si rares que ce que l'on pouvait penser.

D'abord, il y a le contenu de la base de données qui, même s'il est difficile d'en tirer des affirmations, peut confirmer ou infirmer des hypothèses faites sur base d'autres sources d'information.

Dans le cas de certains SGBD, on peut trouver une déclaration du schéma global. Cette déclaration est faite dans un langage propre au SGBD : Le DDL (Data Description Language). Dans le même langage, seront décrites des vues ou sous-schémas. Celles-ci seront, intégrées pour construire le schéma global si celui-ci n'est pas disponible. Dans le cas contraire, elles pourront affiner la perception que l'on a de ce schéma global.

Lors du processus de conception de l'application et plus précisément lors de l'implémentation, toutes les contraintes du schéma logique n'ont pu être exprimées dans le DDL. Mais, si l'implémentation a été faite sérieusement, on pourra retrouver les informations concernant ces contraintes non supportées par le DDL dans les textes sources de l'application (exemple: Conditions avant modification de la base de données), la définition de l'interface (exemple : Conditions d'acceptation d'une saisie) ou des procédures propres au SGBD (exemple : Triggers, check clauses, ...)

Enfin, le schéma physique avec tous ses paramètres peut nous donner des indications précieuses, par exemple la déclaration d'un index n'acceptant pas de données dupliquées nous fera penser à la déclaration d'un identifiant.

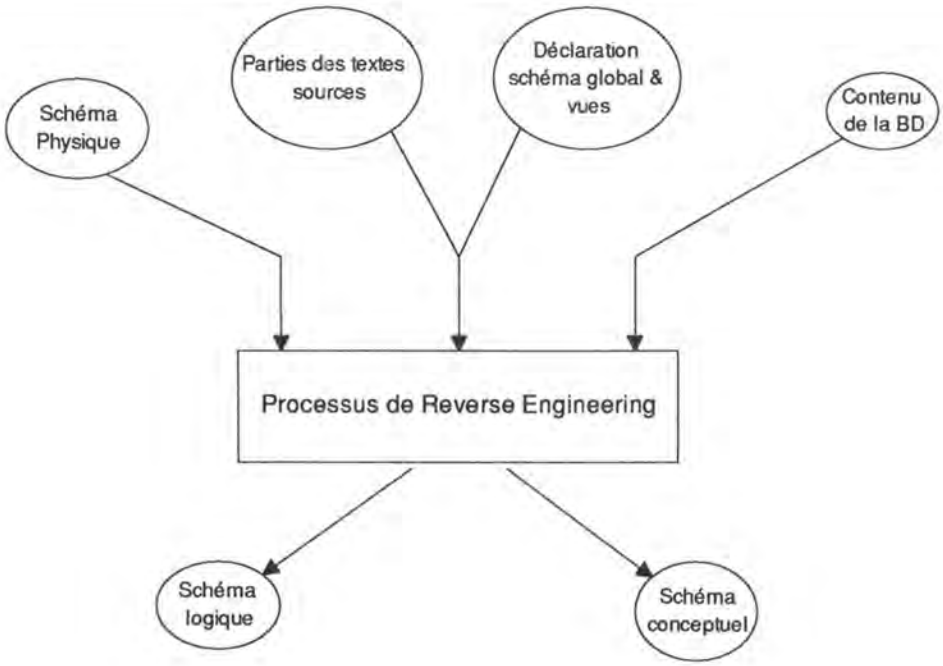


Figure 2.1.

Les résultats attendus d'un processus de rétro-ingénierie dépendent de l'objectif que l'on a en utilisant le ce processus: soit le schéma conceptuel est nécessaire, soit le schéma logique suffit. En effet, pour passer d'une base de données à une autre de la même famille, c'est à dire qui suit le même schéma logique, le schéma conceptuel ne sera pas indispensable mais le schéma logique sera requis.

Il serait peut-être bon, alors, de sectionner le processus de rétro-ingénierie en deux étapes. La première de ces étapes aurait comme résultat le schéma logique de la base de données et la seconde le schéma conceptuel. Il est intéressant de remarquer que ces deux étapes sont exactement l'inverse des phases de conception physique et logique rencontrées dans le processus de forward engineering.

Nous nommerons la première phase "extraction des structures de données" (data structure extraction). Elle sera l'inverse de la conception physique. Tandis que la seconde phase qui est l'inverse de la conception logique se nommera "conceptualisation des structures de données" (data structure conceptualization).

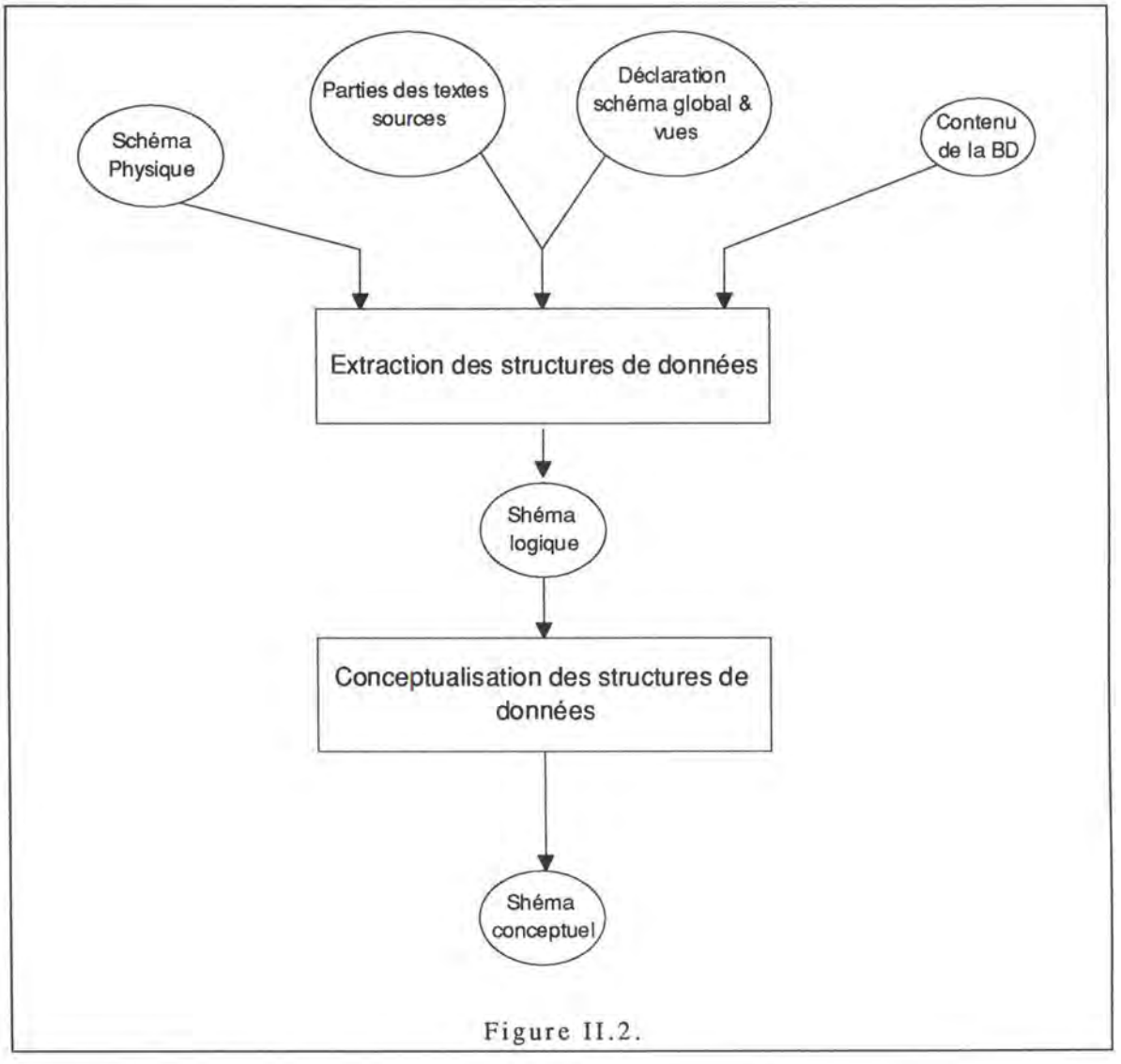


Figure II.2.

2.1.2.1. L'EXTRACTION DES STRUCTURES DE DONNÉES

Comme nous le voyons à la figure II.2. , le but de cette première phase est de fournir une description complète de la structure des données selon le modèle logique de la base de données. De l'ensemble de ses données en entrée, une peut faire défaut : La déclaration du schéma global. Comme nous le décrivons ci-dessous, de cette carence dépendra la stratégie adoptée.

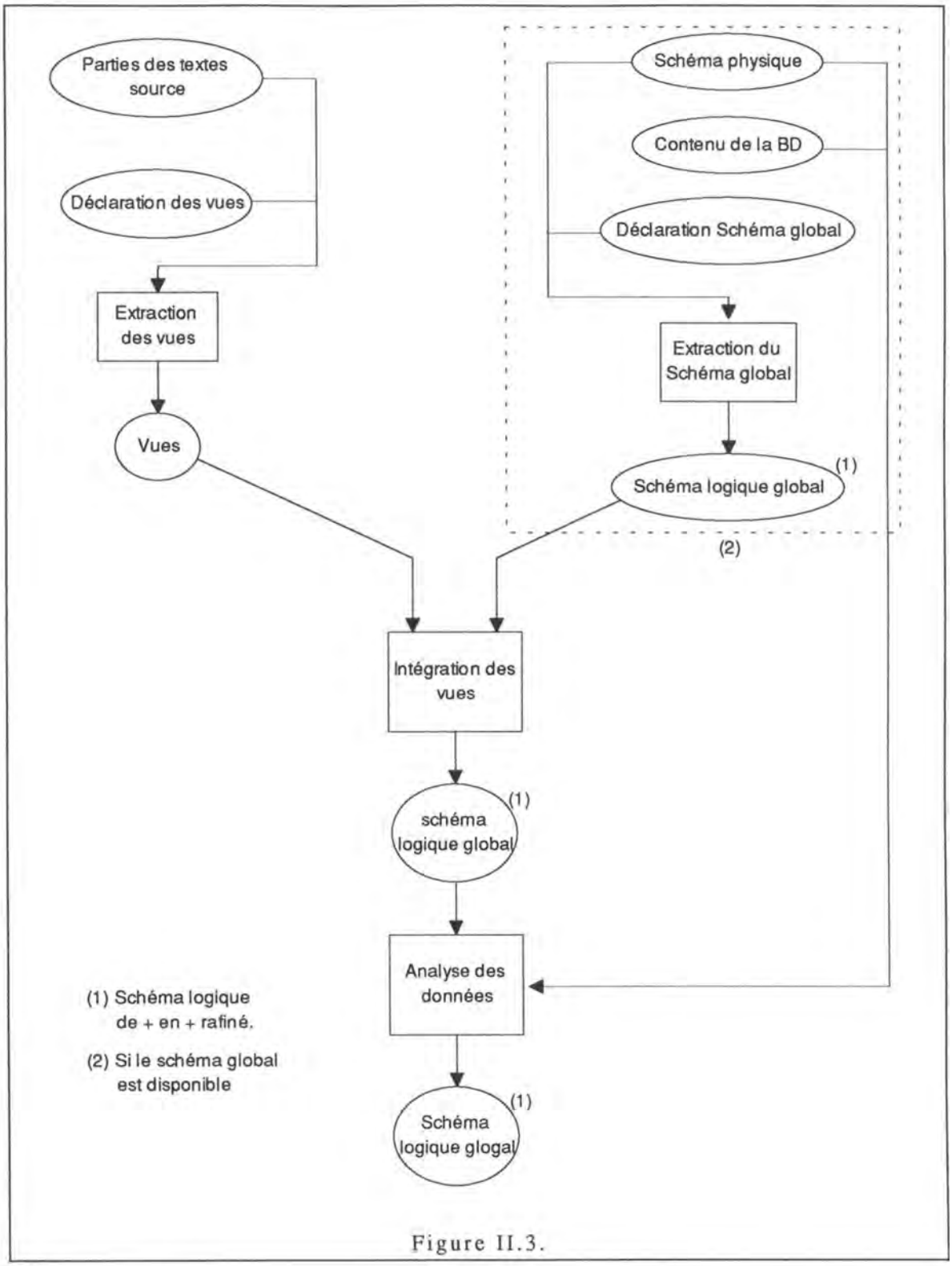

Parties des textes source
Déclaration des vues
Extraction des vues
Vues
Schéma physique
Contenu de la BD
Déclaration Schéma global
Extraction du Schéma global
Schéma logique global
(1)
(2)
Intégration des vues
schéma logique global
(1)
Analyse des données
Schéma logique glogal
(1)
(1) Schéma logique de + en + rafiné.
(2) Si le schéma global est disponible

Figure II.3.

Extraction des vues.

L'extraction des vues a pour objet de retrouver la vision des données qu'ont les programmes de l'application. D'un système à l'autre, une déclaration de cette vision pourra revêtir différentes formes :

1. La déclaration d'une structure de "record" dans la "DATA DIVISION" d'un programme cobol. Cette déclaration peut diverger d'un programme à l'autre.
2. La déclaration d'un "subschema" codasyl dans un programme de l'application. Ici aussi, cette déclaration peut diverger d'un programme à l'autre mais aussi de la déclaration du schéma global.
3. La déclaration d'une vue dans le langage de description des données d'un SGBD relationnel. Cette déclaration n'est rien d'autre que la déclaration d'une table reprenant le résultat d'opérations de lecture effectuées sur les table de bases.

Dans tous ces cas, nous pouvons remarquer que la vue peut être considérée comme doublement partielle. Primo parce qu'elle ne décrit qu'une partie de la base de données et secundo parce qu'elle peut contenir des structures cachées (ex : un filler dans un record cobol) ou des données dérivées d'autres (ex : vues dans un SGBD relationnel contenant la somme des valeurs contenues dans une colonne d'une table.)

En plus de la déclaration des vues, certaines parties des textes sources des programmes les utilisant seront peut-être nécessaires pour affiner la définition de ces vues. C'est le cas, par exemple, dans un texte cobol où on a:

```
...
RECORD SECTION.
01 CLIENT.
   02 ADR_CLI  PIC X(54).
...
WORKING STORAGE SECTION.
01 ADRESSE.
  02 RUE      PIC X(30).
  02 CP       PIC 9(4).
  02 COMMUNE  PIC X(20).
...
PROCEDURE DIVISION.
...
MOVE ADR_CLI TO ADRESSE.
...
```

L'analyse des lignes de la "PROCEDURE DIVISION" et de la "WORKING STORAGE SECTION" permet d'affiner la vue que ce programme a d'un client.

Extraction du schéma global.

Ce processus est relativement simple, il s'agit d'une analyse des sources de la déclaration du schéma global des données. Ce processus interviendra plutôt quand la base de données sera gérée par un SGBD. En plus du schéma global, nous aurons besoin de renseignements d'ordre physique tels que : les déclarations d'index sur certains fichiers qui peuvent indiquer l'existence d'un identifiant.

Intégration des vues.

L'intégration des vues est un problème spécifique de la rétro-ingénierie. C'est pourquoi nous y reviendrons, plus en détail, au chapitre suivant. Mais nous aimerions souligner, ici, la distinction entre le cas où le schéma global est disponible et le cas contraire.

Dans le premier, il s'agit de raffiner le schéma global (exemple: Un attribut de huit caractères devient une date). De plus, pour certain SGBD, ce processus peut donner des indications sur des contraintes qui ne sont pas explicites. Par exemple, une vue dans un SGBD relationnel peut se baser sur une jointure entre deux tables et donc exprimer l'existence d'une relation entre ces deux tables (d'où l'existence d'une clé de référence).

Dans le second cas, il suffira d'élaborer le schéma global par la fusion des sous-schémas. Cette technique sera expliquée dans un chapitre suivant.

Analyse des données

Le contenu de la base de données et le schéma global physique, s'il est disponible, sont analysés afin de raffiner le schéma logique déjà disponible. Par raffiner nous comprendrons trouver des indications ou des confirmations sur l'existence d'un identifiant, la structure d'un attribut, ...

2.1.2.2. LA CONCEPTUALISATION DES STRUCTURES DE DONNÉES

Comme présenté précédemment, ce processus a pour but de transformer le schéma logique, fruit du processus d'extraction, en un schéma conceptuel. Autrement dit et pour revenir à notre modèle unique, ce processus transforme des expressions en termes de concepts techniques en des expressions en termes de concepts abstraits.

Il est à noter que ce processus est l'inverse du processus de conception logique. Ce dernier étant divisé en deux phases, il nous a paru normal de faire de même pour le processus de conceptualisation des structures de données. Nous aurons donc deux sous-processus. Le premier sera dépendant du modèle logique de la base de données et le second en sera indépendant.

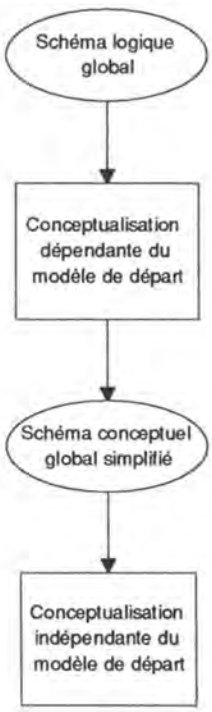


Figure II.4.

2.1.2.2.1. LA CONCEPTUALISATION DÉPENDANTE DU MODÈLE DE DÉPART

Lors de la conception logique, après avoir simplifié le schéma conceptuel, on exécute trois processus :

- Une optimisation du schéma simplifié indépendante du modèle logique cible.
- Une traduction de ce schéma en terme de construction propre au modèle logique cible.
- Une optimisation dépendante du modèle logique cible.

Donc, il parait souhaitable de diviser la phase de conceptualisation dépendante du modèle de départ en trois processus qui chacun serait l'inverse d'un des trois processus ci-dessus. C'est comme cela que nous retrouverons :

- Une "dé-optimisation" du schéma logique dépendante du modèle sur lequel ce schéma repose.
- Une "dé-traduction" de ce schéma en terme de construction de niveau d'abstraction supérieur. Cette phase donnera le schéma conceptuel simplifié.
- Une "dé-optimisation" de ce schéma indépendante du modèle sur lequel il repose.

Nous obtiendrons donc, à la sortie de cette phase, un schéma conceptuel simplifié (c'est à dire sans construction de trop haut niveau d'abstraction telles que les associations plus que binaire, isa, ...) qui ne tient aucunement compte d'aspect d'optimisation.

2.1.2.2.2. LA CONCEPTUALISATION INDÉPENDANTE DU MODÈLE DE DÉPART

Cette phase est l'inverse de la phase de simplification du schéma conceptuel rencontrée lors de la conception logique. C'est ici que l'on cherchera à exprimer les faits et objets avec des constructions de haut niveau d'abstraction.

Cette phase devrait être optionnelle car elle est dépendante du niveau d'abstraction que l'utilisateur souhaite pour le schéma conceptuel résultant du processus de rétro-ingénierie qu'il a demandé.

2.1.3. LES PROBLÈMES SPÉCIFIQUES À LA RÉTRO-INGÉNIERIE

2.1.3.1. L'ANALYSE DES NOMS

Ce chapitre est inspiré de **[Phe]**.

L'analyse des noms est une activité importante dans un processus de rétro-ingénierie. Elle pourra rarement aboutir à des certitudes mais donnera souvent des indications sur les objets de la base de données ou sur les programmes qui la manipulent. On l'utilisera aussi bien dans les transformations de schémas, dans l'intégration de schémas, que dans la réduction de redondance. De plus, elle pourra être utilisée pour donner à ces objets des noms significatifs.

Il est une règle que l'on peut considérer comme générale, même si elle n'est pas absolue : Le programmeur a utilisé des termes significatifs pour nommer les objets. De ce fait, les noms des objets sont porteurs de sémantique. Voyons le genre de renseignements que l'on peut en tirer.

D'abord, le nom d'un objet de la base de données peut être identique à celui de l'élément du monde réel qu'il représente. C'est le cas d'une entité nommée "CLIENT" et qui désigne un client dans le monde réel.

Ensuite, on peut qualifier le nom de l'objet pour indiquer qu'il représente un sous-groupe d'un ensemble d'éléments du monde réel. C'est le cas d'une entité nommée "EMPLOYE-FEMININ".

De plus, on peut indiquer le type d'usage que l'on fait de l'objet. C'est le cas d'un record que l'on garnit temporairement et qu'on appelle "CLIENT-TEMP".

Enfin, on peut aussi, grâce aux noms que l'on donne aux objets, indiquer des relations entre eux. C'est l'exemple de l'entité "COMMANDE" contenant l'attribut "ADRESSE-CLIENT". On voit immédiatement que cet attribut contient des renseignements non pas sur la commande mais sur le client qui a passé cette commande. Une variante de cette analyse se retrouve dans l'intégration de deux schémas lorsque l'on essaye de mettre en relation une construction de chaque schéma. Il s'agit alors de retrouver des similarités entre les noms de ces deux constructions. Une analyse identique se fait lors de la réduction de redondance quand on cherche deux constructions représentant le même objet. La similarité entre deux noms peut se marquer de diverses manières. La notion de similarité de noms étant importante, nous allons indiquer ces différentes similarités.

Le cas le plus simple de similarité de deux noms est celui où ils sont identiques. Mais, on peut en trouver d'autres où les noms, sans être identiques, présentent des similarités. Par exemple, si l'un des deux présente une faute d'orthographe ou s'ils sont identiques à un préfixe (suffixe) près. On peut aussi imaginer le cas où les deux noms sont totalement différents mais veulent dire la même chose dans des langues différentes. On pourrait continuer cette liste qui n'est limitée que par notre imagination.

Nous remarquerons cependant que la similarité des noms de deux objets n'indique pas forcément qu'ils représentent le même élément du monde réel. On ne peut faire, ici, que des suppositions.

2.1.3.2. L'INTÉGRATION DE SCHÉMAS

Ce chapitre est inspiré de **[Phe]** et **[Qui92]**.

2.1.3.2.1. L'INTÉGRATION DE SCHÉMAS EN GÉNÉRAL

L'intégration de schémas est un problème bien connu dans le domaine de la gestion de données. En effet, dans le processus de conception, quand le domaine étudié est trop grand, une vue globale des informations à gérer est impossible à réaliser en une fois. Pour résoudre ce problème, on étudie séparément des sous-domaines (souvent par des personnes différentes) qui donnent des vues locales du réel. Ce n'est qu'après cette étape que l'on procède à une intégration des différents schémas. Cette intégration donne un schéma unique qui représente toutes les informations gérées par la future application. Ce processus d'intégration est vital car la simple addition des schémas produirait une redondance structurelle inacceptable. Cette redondance, dans un processus d'intégration, est éliminée en recherchant les parties des différents schémas qui représentent le même fait réel et en unifiant leurs différentes représentations en une seule.

Le schéma intégré obtenu par un tel processus répondra donc au critère de minimalité puisqu'un même concept apparaissant dans les différents schémas ne sera plus exprimé qu'une seule fois dans le schéma intégré. De plus, le critère de complétude sera également respecté, car chaque concept exprimé dans l'un des sous-schémas sera exprimé dans le schéma intégré. Il n'y aura plus qu'un seul critère auquel il faudra porter une attention toute particulière : le critère de compréhensibilité. Car, n'oublions pas qu'un schéma est avant tout un outil de communication entre différents acteurs.

Mais qui dit "fusionner plusieurs structures de données en une seule" dit aussi "résoudre des conflits". Les types de conflits rencontrés sont au nombre de trois :

- Conflit de description : Un ensemble d'objets du monde réel est décrit de manière différente c'est-à-dire avec des ensembles différents de propriétés.

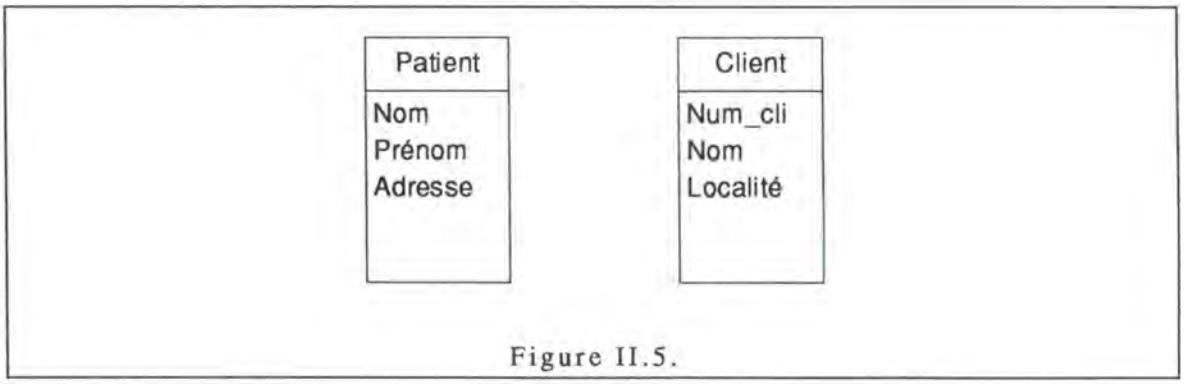


Figure II.5.

Ces deux types d'entité représentent l'ensemble des personnes qui sont clients chez un médecin (donc patients), ces deux représentations ne décrivent pas l'ensemble des objets du monde réel avec les mêmes attributs.

- Conflit structurel : Un même ensemble de faits ou d'objets du monde réel est représenté par des structures de données différentes.

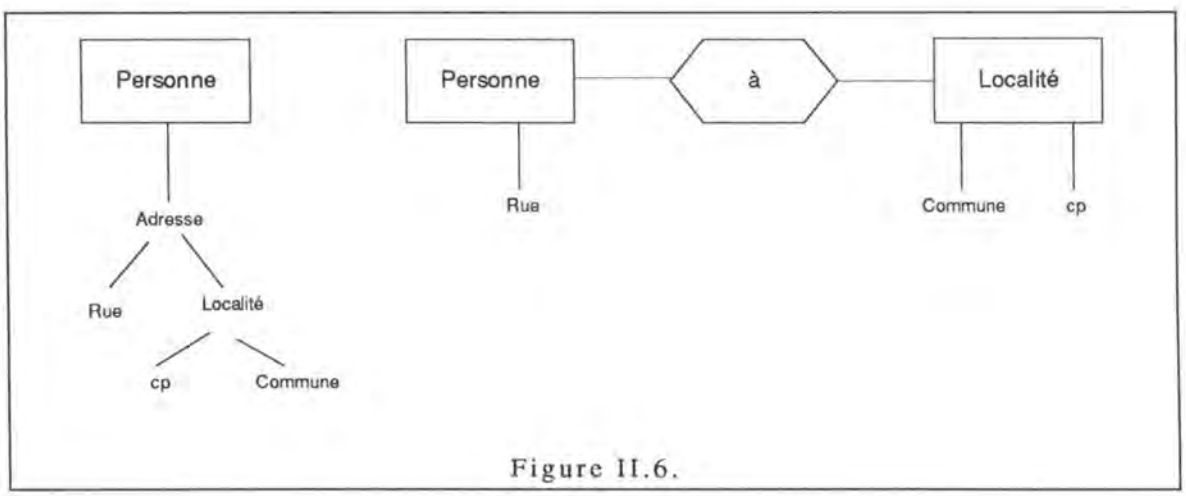


Figure II.6.

- Conflit sémantique : Des schémas adoptent une classification différente pour un même ensemble d'objets du réel.

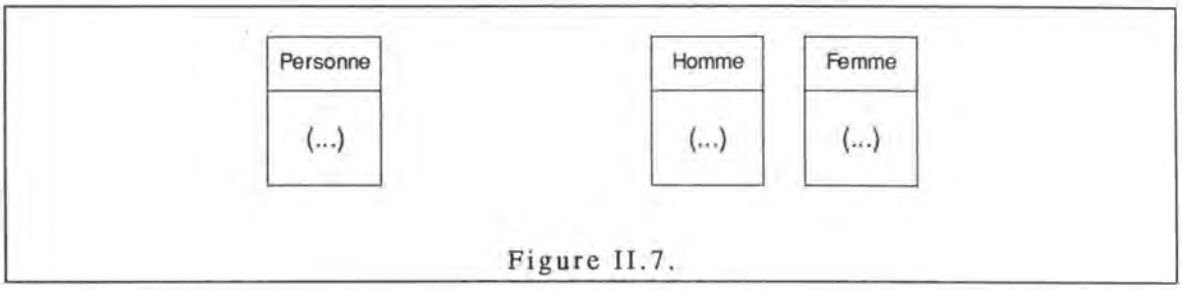


Figure II.7.

On remarquera le **recouvrement** de l'objet "Personne" par les objets "Femmes" et "Hommes"

2.1.3.2.2. LES PARTICULARITÉS DE L'INTÉGRATION DE SCHÉMAS DANS LE CADRE DE LA RÉTRO-INGÉNIERIE

Dans le cadre de la rétro-ingénierie, l'intégration de schémas peut se faire pendant l'une des deux phases du processus ou même après celui-ci. De toute manière, il y a quelques remarques que nous voudrions faire :

La première remarque porte sur les règles de correspondance et d'intégration dans le cas où l'on procède à la fusion de schémas lors de la phase d'extraction des structures de données. Ce choix a comme conséquence que l'on travaille sur des structures techniques incluant des constructions propres à un modèle de base de données et des constructions dont le seul but est d'augmenter les performances de l'application. Il est donc normal d'adapter les règles de correspondance et d'intégration à de telles structures.

De plus, ces règles pourront reposer sur des indications telles que l'arrangement physique des données. Ce type d'indication est propre au fait que l'on travaille sur des descriptions physiques, et non conceptuelles comme ce serait le cas si l'on exécutait l'intégration de schémas au niveau de la phase de conceptualisation des structures de données ou même après le processus de rétro-ingénierie.

Deuxièmement, il est à noter que l'on travail, dans le cadre d'un processus de rétro-ingénierie, sur une base de données réelle et déjà implémentée. Les vues décrivent donc des structures de données cohérentes. Donc, des problèmes du type conflit sémantique n'interviendront pas dans ce cas-ci. Par contre, on pourra trouver des conflits de description (ex: Deux descriptions cobol d'un même record dont une contient un "filler").

2.1.3.2.3. Les étapes et stratégies d'une phase de fusion de schémas

2.1.3.2.3.1. Les étapes

Les étapes d'une phase de fusion de schémas sont au nombre de quatre, la première et la dernière étant optionnelles.

Première étape : La préparation des schémas

La préparation des schémas à intégrer a pour but de faciliter les étapes suivantes. Par exemple, on sait qu'il est plus facile d'intégrer des schémas dont les correspondances sont homogènes (correspondances portant sur des constructions du même type : type d'entité, type d'association, Attribut) que des schémas où les correspondances sont hétérogènes. Donc, un des travaux de l'étape de préparation des schémas à intégrer pourrait être de les restructurer afin qu'un même concept soit représenté dans les différents schémas par un même type de représentation.

Deuxième étape : L'expression des correspondances

Cette étape a pour but d'exprimer toutes les correspondances trouvées par la comparaison des schémas. Cette expression pourrait prendre la forme d'assertion.

Il est à noter que c'est à cette étape que se fait le plus gros du travail de résolution de conflit.

Troisième étape : L'intégration proprement dite

Les objets sont intégrés sur base des correspondances exprimées. Le résultat est un seul schéma intégré.

Quatrième étape : La restructuration du schéma intégré

Cette étape a pour but d'appliquer des critères de compréhensibilité au schéma intégré.

2.1.3.2.3.2. LES STRATÉGIES

La méthode d'intégration que l'on vient d'énoncer peut suivre diverses stratégies.

D'abord, remettons-nous dans le cadre de la rétro-ingénierie. Si on a, à notre disposition, un schéma global, on procédera à son augmentation par fusions successives. Par contre, si on ne possède pas de schéma global, on en créera un à partir de la fusion des sous-schémas. Pour ce faire, on pourrait fusionner tous les sous-schémas en une fois (stratégie n-aire). Mais cette stratégie implique une plus grande complexité du processus de fusion. C'est pour cela qu'on préférera adopter une stratégie binaire, c'est à dire une stratégie où les fusions se font sur base de deux schémas.

2.1.3.3. LA REDONDANCE DES DONNÉES

Ce chapitre est inspiré de **[Phe]**.

Dans la gestion des données, on peut remarquer un invariant : Soit, on optimise la place que ces données occupent et alors on le fait au détriment des performances des traitements qui les manipulent. Soit, on cherche à optimiser ces performances en introduisant de la redondance dans les données.

Par exemple, on peut remarquer dans la figure ci-dessous que l'on a dupliqué l'information concernant l'adresse du client. De ce fait, le traitement qui, par exemple, a pour but d'imprimer les bons de livraison n'aura pas à accéder aux données concernant le client via l'association A. Il y a donc une augmentation de la disponibilité de l'information qui entraîne une augmentation des performances du traitement (moins d'accès disque) mais aussi une augmentation de l'espace disque nécessaire pour stocker l'information.

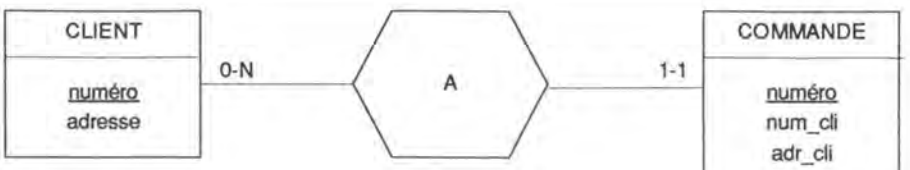


Figure II.8.

Dans le cadre d'un processus de rétro-ingénierie, on devra être capable de réduire cette redondance. C'est à dire, partant d'une base de données qui a "subi les méfaits" de l'optimisation des processus de traitement, il nous faudra retrouver un schéma conceptuel comportant un minimum de redondance. Cette nécessité provient de deux faits :

- Primo: au niveau conceptuel, on ne prend pas en compte les aspects d'optimisation. La qualité primordiale d'un tel schéma est son caractère de représentativité du réel perçu.
- Secundo: dans un processus de rétro-ingénierie, utilisé lors d'une migration d'une base de données à une autre, on remonte justement au niveau conceptuel pour ne pas transporter des optimisations, dédiées à la première, vers la deuxième.

Dans ce chapitre, nous allons expliquer les deux principaux types de redondance : La redondance structurelle et la redondance de dénormalisation.

2.1.3.3.1. La redondance structurelle

Nous pouvons définir ce type de redondance comme ceci : Une construction B est redondante avec une construction A si, et seulement si, quel que soit l'état de la base de données, chaque instance de B peut être dérivée d'une instance de A.

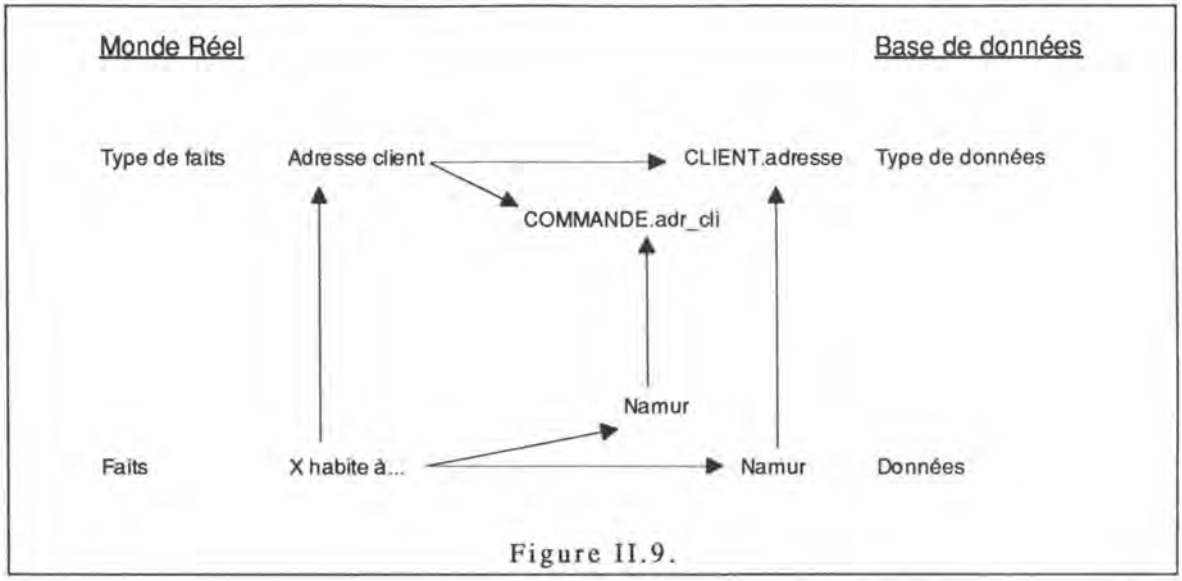


Figure II.9.

Comme nous le voyons dans la figure ci-dessus, la valeur de l'article "COMMANDE.adr_cli" peut toujours être dérivée à partir de l'entité "CLIENT" attachée à la commande. Ce type de redondance est visible sur le schéma de la base de données. Pour éliminer cette redondance, il suffit d'éliminer la structure redondante c'est à dire l'attribut "COMMANDE.adr_cli".

Dans le cas où la redondance serait symétrique, c'est à dire si la construction B est redondante à la construction A et réciproquement (ce serait le cas si la cardinalité du rôle joué par CLIENT était 1-N), le choix de la construction à éliminer serait arbitraire. Mais on préférera garder la construction la plus générale. On gardera donc le type d'entité plutôt que le type d'association et ce dernier plutôt que l'attribut.

2.1.3.3.2. La redondance de dénormalisation

Ce type de redondance n'est visible qu'au niveau des instances des données et non au niveau du schéma de la base de données comme c'est le cas pour la redondance structurelle. Il s'agit d'une redondance qui provient du fait que, dans l'ensemble des instances d'une construction, un même fait est exprimé plusieurs fois. Si nous regardons uniquement le type d'entité CLIENT de la figure II.10. nous n'apercevrons aucune redondance. Pourtant, le fait que le client X habite à Namur sera représenté chaque fois qu'il aura passé une commande.

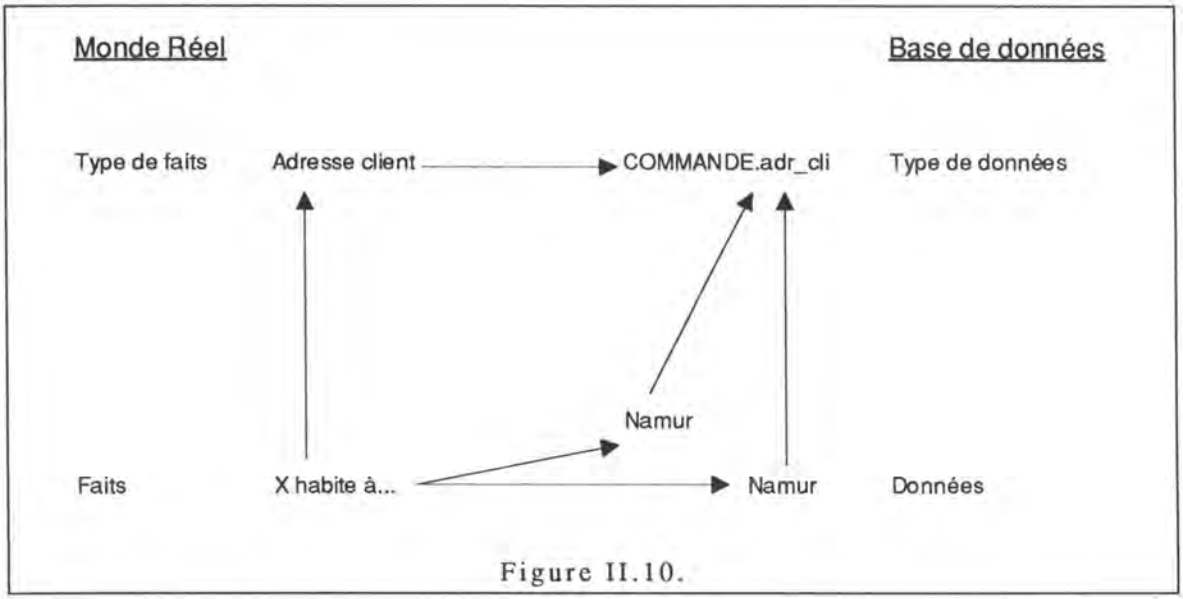


Figure II.10.

Dans le modèle relationnel, une redondance de ce type est éliminée en normalisant le schéma. C'est à dire que la théorie relationnelle nous fournit un critère pour éviter ce type de redondance : "Tous les attributs doivent dépendre fonctionnellement et uniquement d'un identifiant." (Troisième forme normale de Boyce-Codd). Ce critère peut s'appliquer facilement à l'expression relationnelle d'un schéma entité-association.

2.1.3.3.3. REMARQUES

Nous voudrions faire deux remarques sur ce qui vient d'être dit:

- Primo: comme nous l'avons vu dans l'exemple qui illustrait ce chapitre, une même construction peut être à la base de redondances des deux types.
- Secundo: le plus gros problème, qui se pose dans un processus de rétro-ingénierie, n'est pas la réduction de la redondance mais la détection de cette redondance. Dans le cas de la redondance structurelle, on pourra, par exemple, se baser sur l'analyse des noms car, les structures redondantes ont souvent des noms proches. Tandis que dans le cas de la redondance de dénormalisation, il faudra se baser sur l'analyse de dépendances fonctionnelles.

2.1.3.4. LES TRANSFORMATIONS DE SCHÉMAS

Ce chapitre n'a pas pour but de présenter les différentes transformations de schémas nécessaires à un processus de rétro-ingénierie. Celles-ci seront décrites en détail dans les parties suivantes du présent document. Ici, nous allons simplement définir ce qu'est une transformation et situer leurs utilisations dans le processus de rétro-ingénierie.

Une transformation T sur un schéma S formé de constructions de représentation de données est le remplacement d'une de ces constructions par une autre, qui est sémantiquement équivalente, pour former un nouveau schéma S'. L'équivalence sémantique est nécessaire si l'on veut exprimer dans S' tout ce que l'on pouvait exprimer dans S. De plus, elle est la condition nécessaire et suffisante pour que la transformation soit réversible, c'est à dire pour qu'il soit possible de trouver une transformation T^{-1}, inverse de T, transformant S' en S. Il est à noter que la composition de deux transformations réversibles est elle-même réversible. Cette remarque nous paraît importante car la composition de transformation sera souvent utilisée.

Nous remarquerons également que le schéma S pourra devoir répondre à une précondition pour que la transformation T puisse lui être appliquée. De même, la transformation T pourra donner naissance à une postcondition à laquelle le schéma S' répondra.

La figure suivante (reprise de **[Hai86]**) contient :

- Une précondition qui exprime que tout article COMMANDE en relation avec un article PRODUIT via la relation CP doit être en relation avec un article COLIS via la relation CC pour que la transformation soit applicable.
- Une postcondition qui exprime qu'après l'application de la transformation, tout article COLIS en relation avec un article PRODUIT via la relation PC sera en relation avec un article COMMANDE via la relation CC.

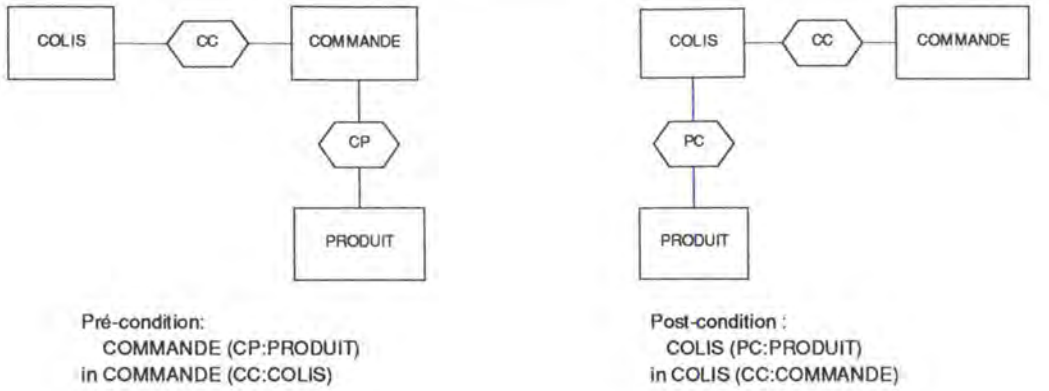


Figure II.11.

Dans un processus de rétro-ingénierie, les transformations seront utilisées la plupart du temps dans la phase de conceptualisation des structures de données pour remplacer des constructions par d'autres d'un niveau d'abstraction supérieur ou identique mais plus explicite.

Les deux parties qui suivent sont des traitements à affecter au contenu de la base de données, indépendement de tout SGBD. Ils pourraient avoir comme source, un simple fichier ASCII reprenant les données de la base.

2.1.3.5. TYPE ET STRUCTURES RÉELS

Nous désignerons type réel d'une donnée, le domaine des valeurs prises par cette donnée dans la base de données. La recherche des domaines et des structures peut s'avérer utile losqu'il s'agit de rechercher des attributs de référence au sein d'une entité, ou de donner une définition sémantique à un attribut sur base de sa structure.

Pour la recherche d'attributs de référence, le critère pris en compte, sera que les deux attributs (ou groupes d'attributs) concernés doivent avoir un domaine compatible, c'est à dire que l'ensemble des valeurs de l'un doit-être inclus dans l'ensemble des valeurs de l'autre.

La structure des données sera surtout un indice pour l'opérateur, lui permettant de placer une définition plausible sur ce qu'est la donnée et son utilisation éventuelle.

Cette analyse s'occupe surtout de l'étude du domaine des données de types communs tels que les chaînes de caractère ou les nombres.

fonctionnement:

Considérons un attribut dont le domaine de définition dans la base de donnée, peut être ramené à une longueur n.

La structure sera représentée par un vecteur de n entrées. Pour chaque entrée on a les informations suivantes :

tab[i] = {val_min,val_max,type} (1<=i<=n) .

où

tab[i] est l'ensemble des informations correspondant à la $i^{ième}$ position de l'attribut,

val_min est la valeur minimale du caractère pour cette position,

val_max, la valeur maximale,

Type est le type réel pour cette position.

Le vecteur de n triplets est initialisé avec val_min=255, val_max=0 et type = ?. Chaque occurrence de l'attribut est découpée en n parties qui sont examinées séparément, afin de mettre à jour le triplet correspondant du type réel.

Les types réels sont déterminés de manière à regrouper au mieux les caractères ASCII par leur utilisation. Les différents types réels tels que nous les avons défini sont les suivants :

- Type N : **N**umérique, les caractères de 0 à 9.
- Type m : Caractères alphabétiques **m**inuscules.
- Type M : Caractères alphabétiques **M**ajuscules.
- Type X : Caractères alphabétiques (M ∪ m).
- Type A : Caractères **A**lphanumériques (X ∪ N).
- Type O : Les "**O**pérateurs" (ex : * - / + . ,).
- Type H : Caractères mathémathiques (O ∪ N).
- Type S : Caractères spéciaux (H ∪ E).
- Type E : Signes séparateurs (ex : & $ \ # ...).
- Type C : Les caractères **C**ourants.
- Type T : Tous les autres caractères.
- Type Z : Tout caractère ASCII.

Exemple : Soit l'attribut NUM_TEL dont le type définit dans la base de données est CHAR(12), et ses occurrences

082/12.23.34 , 061/23.65.95 , 081/73.83.01

L'analyse fournira le résultat suivant :

Attribut NUM_TEL :

{(0,0,N), (6,8,N), (/,/,S), (1,7,N), (2,3,N), (.,.,S), (2,8,N), (3,5,N), (.,.,S), (0,9,N), (1,5,N)}.

Structure réelle : NN'/'NN'.'NN'.'NN

Figure II.12.

Les conversions suivantes sont effectuées lors de l'analyse :

	?	N	m	M	X	A	O	H	S	E	C	T	Z
N	N	N	A	A	A	A	H	H	Z	Z	C	Z	Z
m	m	A	m	X	X	A	Z	Z	C	Z	C	Z	Z
M	M	A	X	M	X	A	Z	Z	C	Z	C	Z	Z
X	-	A	X	X	X	A	Z	Z	C	Z	C	Z	Z
A	-	A	A	A	A	A	Z	Z	C	Z	C	Z	Z
O	O	H	Z	Z	Z	Z	O	H	Z	Z	Z	Z	Z
H	-	H	Z	Z	Z	Z	H	H	S	S	Z	Z	Z
S	-	Z	C	C	C	C	Z	Z	S	S	C	Z	Z
E	E	Z	Z	Z	Z	Z	Z	S	S	E	Z	Z	Z
C	C	C	C	C	C	C	Z	Z	C	Z	C	Z	Z
T	T	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	T	Z
Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z

Tableau II.13.

Les colonnes représentent l'état d'une position de la structure. En ligne, nous avons le type de caractère en cours d'examen. Si, par exemple le type courant de la position est N (numérique), et que le caractère examiné est un '%' de type T, le résultat sera l'intersection de la colonne N et de la ligne T, donc Z.

Le résultat de cette analyse est triple :

- domaine de valeur des attributs,
- type réel,
- Structure réel.

Un second type d'analyse pourrait se révéler utile. Il s'agirait de repérer des valeurs particulières d'attributs, telles que la répétition d'un même caractère, ou une valeur commune à plusieurs d'entre-eux. Le but de cette étude serait de mettre en évidence d'éventuelles valeurs par défaut, ou des représentations de la valeur nulle.

2.1.3.5. DÉPENDANCES FONCTIONNELLES ET IDENTIFIANTS

La recherche de dépendances fonctionnelles (DF) par l'analyse des données est un procédé nécessitant beaucoup de ressources. C'est une analyse exhaustive sur toutes les instances des attributs d'un type d'entité.

Les résultats des algorithmes qui suivent, ne sont que des indices dont l'opérateur pourra tenir compte dans la réorganisation des types d'entité. En effet, Aucune affirmation ne peut être retirée d'un tel examen, car l'existence d'une DF, peut n'être que le fruit du hasard, et/ou la conséquence d'un trop petit nombre d'instances pour le type d'entité.

2.1.3.5.1. LES DÉPENDANCES FONCTIONNELLES.

Soient a et b les attributs du type d'entité E.

Une DF de a vers b, fait correspondre à chaque valeur de a, au plus une valeur de b. Il faut donc examiner chaque couple (a,b), d'après l'algorithme suivant :

Soit V, l'ensemble des instances des couples (a,b) déjà examinées.

```
Algo dependances fonctionnelles
V = {vide}
fini = faux
while ((fini = faux) and (E not = {vide})) do :
    Obtenir (a,b) dans E.
    If (a in V)
        then if (b not in V)
                    fini := vrai
             end-if
        else ajout (a,b) in V
    end-if
end-while
```

Figure II.14.

La procédure Obtenir (a,b) se charge de lire séquentiellement une instance de couple (a,b) du type d'entité E. E est considéré comme vide si tous les couples d'instances de E ont été examinés.

Si la valeur de a, a déjà été trouvée dans un autre couple, tel que (a,b) et (a',b'), avec a=a' et (a',b') ∈ V), alors la valeur de b est connue, et **doit** être b', dépendance fonctionnelle oblige. Si ce n'est pas le cas, on peut affirmer dès cet instant qu'il n'y a pas de DF de a vers b. Il faut alors permuter les (a,b) pour examiner (b,a). Si le type d'entité possède n instances, il faut en moyenne (n/2) examens pour obtenir un résultat. La recherche pour i attributs dans un type d'entité ayant n instances, prend en moyenne :

$$\frac{i * (i-1)}{2!} \cdot \frac{n}{2} \text{ Examens}$$

Cet algorithme simplifié ne considère les DF qu'entre deux attributs. La version suivante, considère le type d'entité dans son entièreté.

Soient a,b,c,d les attributs de E, avec a,b,c,d non identifiants.

L'algorithme se déroule en 3 passes, soit une de moins que le nombre d'attributs.

La figure ci-après représente l'arbre des permutations des attributs. Les noeuds sont les groupes d'attributs à l'origine d'une éventuelle DF. L'algorithme explore l'arbre de gauche à droite et de bas en haut.

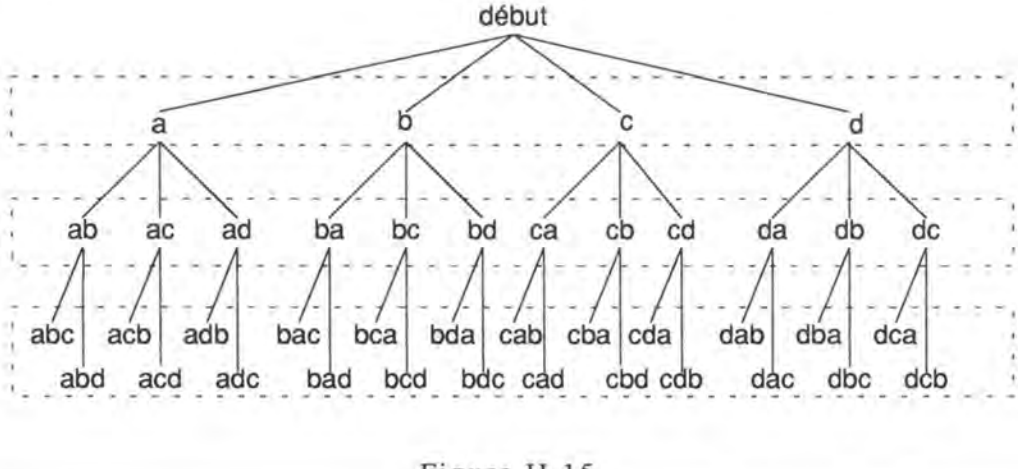


Figure II.15.

L'exploration des noeuds de l'arbre correspondant à un type d'entité ayant beaucoup d'attributs, est une tâche requérant beaucoup de temps. Les moyens de raccourcir ce temps de calcul sont de supprimer les noeuds étant la permutation d'un noeud déjà exploré ou contenant un groupe d'attribut déjà à l'origine d'une DF (donc les identifiants).

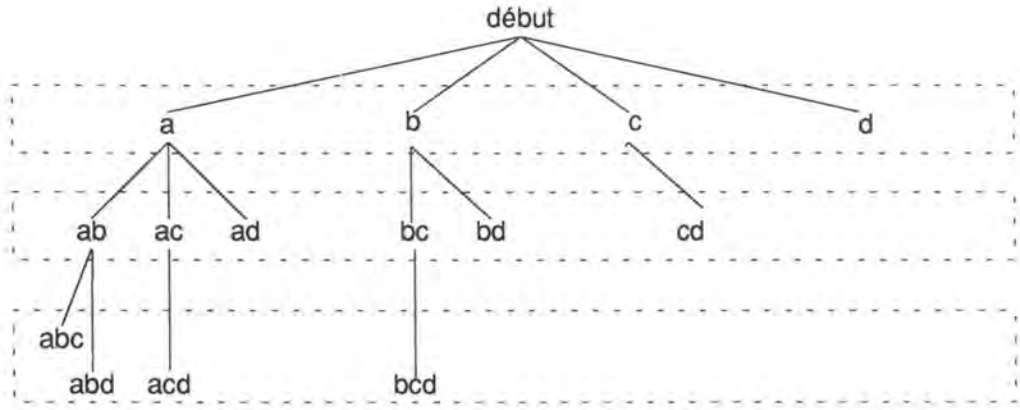


Figure II.16.

Ainsi, si en cours d'examen on constate que le noeud a est à l'origine d'une DF, on évitera d'examiner les noeuds contenant a.

la complexité de l'algorithme se calcule comme suit.

Niveau 1: les 4 attributs.

Niveau 2: les combinaisons des 4 attributs pris par groupe de 2.

Niveau 3: les combinaisons des 4 attributs pris par groupe de 3.

D'une manière générale, pour n attributs, au $i^{ième}$ niveau, il faut prendre le nombre de combinaisons des n attributs pris par groupes de i, avec $1 <= i <= n-1$.

La formule générale, qui établit le nombre d'itérations pour l'examen des n attributs est représentée à la figure II.17.

$$\sum_{i=1}^{n-1} nCi \qquad nCi = \frac{\prod_{j=n-i+1}^{n} j}{i!}$$

Figure II.17.

Le résultat de l'algorithme est un ensemble de couples origine-destination, qui définit les dépendances fonctionnelles dans le type d'entité.

Exemple :

a,b $\Rightarrow$ c

d $\Rightarrow$ e

...

Notons toutefois, que quel que soit le SGBD utilisé, on peut toujours exporter les données d'une base vers un fichier asci, ré-importer ce fichiuer dans une autre base relationnelle et effectuer l'algorithme par une requête dont la forme générale est :

```
SELECT a,COUNT(b)
FROM table
GROUP BY a;
```

Si toutes les valeurs de la colonne COUNT(b) sont égales à "1", alors, nous sommes en présence d'une dépendance fonctionnelle de a vers b.

2.1.3.5.2. LES IDENTIFIANTS.

Nous proposons 3 manières de rechercher les identifiants d'un type d'entité à partir des dépendances fonctionnelles.

1) Un identifiant, est un attribut ou un groupe d'attributs, qui est à l'origine d'une DF vers tous les autres attributs du type d'entité. Les identifiants d'un type d'entité, peuvent donc être obtenus par une variante de l'algorithme de recherche des dépendances fonctionnelles. Dans cet algorithme, l'ensemble d'attributs cible de la DF, est constitué de tous les attributs du type d'entité ne figurant pas dans le groupe source de la DF.

Pour plus d'information sur les deux méthodes qui suivent, et que nous ne développerons pas ici, nous renvoyons le lecteur à la consultation de **[Hai89]**.

Ces deux techniques se basent sur les dépendances fonctionnelles entres attributs. Elles peuvent donc utiliser le résultat de l'algorithme de recherche des DF au sein d'un type d'entité.

2) Détection des identifiants - première technique (4.7).

1. On choisit un déterminant qui ne contient pas de déterminé (ou à défaut un déterminant). Ce déterminant constitue la proposition initiale d'identifiant.
2. En consultant la fermeture transitive, on repère les attributs ainsi déterminés.
3. S'il reste des attributs non encore déterminés, on ajoute à la proposition d'identifiant l'un d'entre eux qui n'est déterminé dans aucune DF (ou à défaut l'un quelconque d'entre eux) et on reprend l'analyse au point deux.
4. La proposition d'identifiant est un identifiant. Si plus d'un identifiant a déjà été trouvé, on vérifie qu'aucun n'est un sous-ensemble d'un autre. Dans ce cas, ce dernier serait non strict et devrait être éliminé.
5. Si dans les points 1 et 3, plusieurs solutions étaient possibles, on examinera les autres possibilités (donc retour au point 3, puis au point 1).

3) Détection des identifiants - seconde technique (4.8).

La relation peut posséder plus d'un identifiant.

1. l'identifiant de départ est l'ensemble des attributs de la relation. Appelons-le **I**.
2. Rechercher le noyau irréductible **N**. C'est à dire un noyau ne contenant plus d'attribut qui soit à la fois déterminé et non déterminant.
3. Pour chaque **DF** (appelée df_i) du noyau **N**,
   3.1 Enlever dfi de **N**, ce qui donne **I**i
   3.2 Rechercher $\mathbf{N}_i$, le noyau irréductible de $\mathbf{I}_i$
4. Les identifiants sont les ensembles $\mathbf{N}_i$ qui ne sont sur-ensemble d'aucun autre.

Nous tenons à insister sur le fait que les résultats de ces recherches ne peuvent être utilisés qu'à titre indicatif, et qu'ils ne contiennent aucune affirmation.

3. LA RÉTRO-INGÉNIERIE D'UNE BASE DE DONNÉES RELATIONNELLE

3.1. LES BASES DE DONNÉES RELATIONNELLES

3.1.1. LE MODÈLE RELATIONNEL

3.1.1.1. LA PROVENANCE DU MODÈLE

Le modèle relationnel a été proposé par C.F. Codd dans son article "A Relationnal Model of Large Shared Data Bank" publié le 6 juin 1970. Dans cet article, il présente le modèle relationnel en s'intéressant surtout à la représentation de l'information. Il n'y tient pas compte de la technique informatique, de ses exigences et contraintes.

Le modèle relationnel est basé sur un concept à la fois simple et rigoureux : La notion mathématique de relation. C'est là la force de ce modèle qui représentera donc la réalité de manière simple et formelle.

Le modèle relationnel peut se diviser en trois parties : Une partie structurelle reprenant les concepts de base, une partie composée de deux règles d'intégrité qui s'appliquent aux relations et aux attributs référentiels et, enfin, une partie manipulatrice reprenant les différents opérateurs applicables aux concepts de base.

Les deux premières parties permettent une description simple sous forme tabulaire des types d'entité et des types d'association. La troisième, elle, permet une manipulation non procédurale des données. Celle-ci a entraîné le développement de langages de requêtes plus proche de l'utilisateur et donc plus facile à utiliser. Mais notre réflexion se basant uniquement sur les structures de données, nous ne nous attarderons pas sur cette troisième partie.

3.1.1.2. LES CONCEPTS DE BASE

Nous présentons, ici, un schéma reprenant la plupart des concepts de base du modèle relationnel que nous détaillerons en suite.

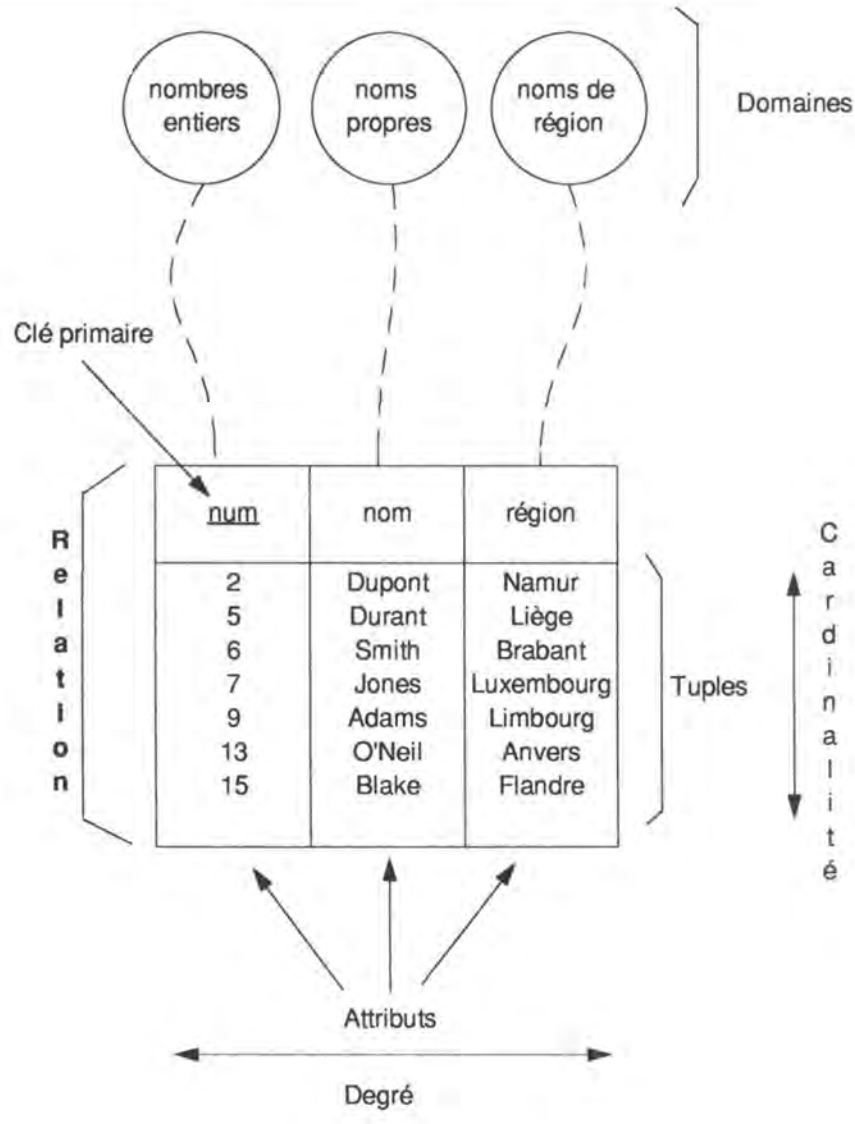


Figure III.1.

3.1.1.2.1. LE CONCEPT DE DOMAINE SIMPLE

Un domaine simple est un ensemble de valeurs atomiques de même type, admissibles par un composant de la relation. Par atomique, nous entendons que ces valeurs ne peuvent pas posséder une structure interne; en d'autres mots, elles ne peuvent pas être décomposables. De ce fait, on peut dire qu'une valeur appartenant à un domaine simple est la plus petite unité sémantique d'information.

Comme tout ensemble, un domaine simple peut être défini en extension ou en compréhension. De plus, le domaine simple sera identifié par un nom.

3.1.1.2.2. LE CONCEPT D'ATTRIBUT

Un attribut défini sur un domaine représente la participation de ce domaine à une relation comme y jouant un rôle. Nous pouvons donc dire que l'attribut sert à déterminer le rôle que joue un domaine dans une relation.

Le nom de l'attribut et le nom du domaine sur lequel il est défini seront souvent identiques. Mais si un domaine joue plus d'un rôle dans une relation, alors les attributs qui seront définis sur lui devront porter des noms différents.

Un attribut étant défini sur un domaine, les valeurs prises par cet attribut doivent appartenir à ce domaine.

3.1.1.2.3. LE CONCEPT DE RELATION

Pour ce concept, nous allons reprendre la définition donnée dans **[Bat92]**.

Une relation R définie sur une collection de domaines D_1, D_2, ..., D_N non nécessairement disjoints, consiste en un en-tête et un corps.

L'en-tête est constituée d'un ensemble de pairs attribut-domaine $\{ (A_1;D_1), (A_2;D_2), ..., (A_n;D_n) \}$ où chaque attribut A_j est défini sur le domaine D_k. La valeur n sera appelée "le degré de la relation".

Le corps est constitué d'un ensemble de tuples variant dans le temps. Chaque tuple est constituée d'un ensemble de pairs attribut-valeur
$\{ (A_1;v_{i1}), (A_2;v_{i2}), ..., (A_n;v_{in}) \}$ où "i" varie, dans une relation, de 1 à m. La valeur m est appelée "la cardinalité de la relation" et indique le nombre de tuples de celle-ci.

Dans chaque tuple, il y a une pair attribut-valeur $(A_k;v_{ik})$ pour chaque attribut A_k de l'en-tête. De plus, pour chaque pair attribut-valeur $(A_k;v_{ik})$, v_{ik} est une valeur du domaine D_k associé à l'attribut A_k

Nous terminerons la description du concept de relation en énonçant quatre règles à son propos :

- Il n'y a pas, dans une relation, deux tuples identiques (qui possèdent les mêmes valeurs pour les mêmes attributs).
- Toutes les valeurs associées à un attribut sont atomiques.
- Il n'y a pas de notion d'ordre dans les tuples.
- Il n'y a pas de notion d'ordre dans les attributs.

3.1.1.2.4. Le concept de clé primaire

Dans le modèle relationnel, le concept de clé est défini comme l'équivalent du concept d'identifiant dans le modèle Entité-Association. Une clé d'une relation est un ensemble d'attributs de la relation qui identifie chaque tuple de toute extension de la relation.

En général, une relation peut avoir plus d'une clé. On les appellera clés candidates. Une de celles-ci sera désignée pour jouer le rôle de clé primaire (ou d'identifiant).

3.1.1.2.5. LE CONCEPT DE CLÉ ÉTRANGÈRE

Ce concept n'aurait su être représenté dans la figure IV.1, car il se base sur un lien entre deux relations R_1 et R_2. En effet, un ensemble d'attributs FK dans la relation R_1 est une clé étrangère (foreign key) faisant référence à une relation R_2 si les valeurs des attributs de FK font référence aux valeurs des attributs de la clé primaire de la relation R_2.

3.1.1.3. LES CONTRAINTES D'INTÉGRITÉ

Une contrainte d'intégrité (CI) est une propriété du monde réel que doivent satisfaire les données contenues dans les extensions d'un schéma.

Nous noterons deux types différents de CI : Celles qui sont définies par l'utilisateur et qui portent sur un schéma spécifique et celles qui sont propres à tout schéma. Ces dernières, que nous allons détailler, sont au nombre de deux : la contrainte de relation et la contrainte de référence.

3.1.1.3.1. LA CONTRAINTE DE RELATION

La contrainte de relation, comme son nom l'indique, s'applique à une relation prise séparément des autres. Plus précisément, cette CI porte sur les valeurs que peut prendre une clé candidate (et donc une clé primaire) dans une relation et s'énonce comme suit :

Dans une relation, la valeur d'une clé candidate pour un tuple donné doit toujours être non nulle et différente des valeurs de cette même clé pour les autres tuples. Dans le cas où la clé candidate est formée de plusieurs attributs, aucun d'entre eux ne pourra se voir assigner la valeur nulle.

Le modèle relationnel étant basé sur la notion d'ensemble, une relation aura toujours une clé candidate : la totalité de ses attributs. De ce fait, dans une relation il n'y aura jamais deux tuples identiques.

Pour de nombreux auteurs, cette contrainte peut se scinder en deux : une contrainte d'unicité et une contrainte de non-nullité.

3.1.1.3.2. La contrainte de référence

En ce qui concerne la contrainte de référence, elle concerne les valeurs prises par une clé étrangère dans une relation R_1 par rapport aux valeurs prises par la clé primaire dans une relation R_2 qu'elle référence. Ici, donc, la CI ne portera plus uniquement sur une relation mais sur le lien entre deux relations.

On peut énoncer informellement cette contrainte comme suit :

Dans une relation, un tuple qui fait référence à une autre relation, via une clé étrangère, doit faire référence à un tuple qui existe dans cette autre relation.

Plus formellement, la condition, donnée ci-dessous, pour une clé étrangère spécifie une CI référentielle entre deux relations R_1 et R_2.

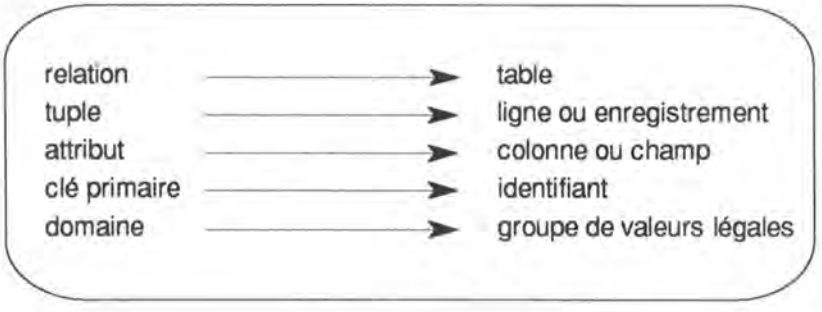
Une clé étrangère dans un tuple t_1 de la relation R_1 possède soit la valeur "null" soit la valeur de la clé primaire pour le tuple t_2 dans la relation R_2.

3.1.2. La mise en pratique du modèle

Nous allons aborder dans ce chapitre les bases de données qui suivent le modèle relationnel. Quand nous disons "suivre", c'est un bien grand mot. Il serait préférable de parler de base de données qui s'inspirent du modèle relationnel.

De fait, le modèle n'est pas toujours rigoureusement respecté par ces bases de données. Nous allons reprendre les concepts de base, les règles d'intégrité et les opérateurs pour montrer la divergence entre le modèle et son application.

En ce qui concerne les concepts de base, le modèle est bien respecté. Indubitablement, ils sont tous repris. La seule remarque à faire est le changement possible de la nomenclature. Ces changements sont repris ci-dessous.



Il n'en va pas de même en ce qui concerne les deux contraintes d'intégrité qui devraient être propres à toutes bases de données relationnelles. Il ne s'agit pas, ici, d'une carence des bases de données elles-mêmes mais plutôt de leur système de gestion.

Du fait qu'il est possible d'insérer deux lignes identiques dans une table, la contrainte de relation n'est pas respectée puisque l'ensemble des attributs de la table ne forme pas une clé candidate. De plus, rares sont les SGBD relationnels qui supportent entièrement la contrainte de référence.

Enfin, en ce qui concerne le langage SQL qui est censé reprendre les opérateurs relationnels, on peut constater qu'il est souvent beaucoup plus riche que l'algèbre relationnelle. Pour seul exemple, nous prendrons la possibilité qui est offerte à l'utilisateur de grouper ou d'ordonner le résultat d'une requête. Or, cette notion d'ordre n'existe pas dans le modèle relationnel puisqu'il se base sur la notion mathématique d'ensemble.

Dans ce contexte, on peut se demander s'il n'existe pas une certaine classification des produits commercialisés qui se disent relationnels. A ce propos, Codd a défini douze règles que doit suivre une base de données (et son système de gestion) pour pouvoir être considérée comme relationnelle. Ces règles sont reprises et expliquées dans **[Dat5]** comme suit :

Règle n° 1 : La règle de l'information

Toute information dans la base de données doit être représentée d'une seule et unique manière, à savoir une valeur dans une colonne et sur une ligne d'une table.

Règle n° 2 : La garantie d'accès

Toute valeur individuelle dans une base de données doit être adressable logiquement en spécifiant les noms de la table et de la colonne qui la contient ainsi que la valeur de la clé primaire de la ligne où elle se trouve.

Règle n° 3 : Le traitement de la valeur "null"

Le SGBD doit pouvoir supporter une représentation de l'"information manquante" et l'"information non applicable" qui soit distincte de toutes autres valeurs possibles de l'attribut et indépendante du domaine sur lequel ce dernier est défini.

Règle n° 4 : Le catalogue en ligne basé sur le modèle relationnel

Le système doit supporter un catalogue dans lequel la description de la base de données est représentée au niveau logique sous forme de tables tout comme les données ordinaires. De plus ces tables devront être accessibles, aux utilisateurs autorisés, via le langage de requête.

Règle n° 5 : Le langage complet

Le système doit supporter un langage relationnel :

- Dont la syntaxe soit définie,
- qui puisse être utilisé interactivement ou dans une application et
- qui contient le langage de définition des données, le langage de manipulation des données et le langage de contrôle des données.

Règle n° 6 : Mise à jour des vues

Toutes les vues qui peuvent théoriquement être mises à jour, doivent l'être par le système. En d'autres mots, si on peut effectuer une mise à jour sur une vue, le système doit assurer la répercussion sur les tables de base.

Règle n° 7 : Insertion, mise à jour et effacement

Le système doit supporter les opérations d'insertion, de mise à jour et d'effacement.

Règle n° 8 : Indépendance des applications vis à vis du niveau physique

Règle n° 9 : Indépendance des applications vis à vis du niveau logique

Règle n° 10 : CI indépendante des applications

Les CI doivent pouvoir être définies dans le langage relationnel et être contenues dans le catalogue. De ce fait, il doit être possible de changer ces contraintes sans devoir effectuer des transformations dans les applications.

Règle n° 11 : Indépendance vis à vis des données distribuées

Quand la base de données est distribuée entre différentes applications, celles-ci n'ont pas de gestion, spécifique à la distribution des données, à réaliser. En d'autre terme, les problèmes dûs à la distribution des données (accès concurrentiel, ...) sont pris en charge par le SGBD.

Règle n° 12 : Non-subversion des CI

Si le SGBD fournit une interface de bas niveau, celui-ci ne peut être utilisé pour passer outre des CI exprimées dans l'interface de haut niveau.

3.2. LES SOURCES D'INFORMATION DANS LES SGBD RELATIONNELS

Les descriptions suivantes sont tirés divers SGBD dont **[Syb]** et **[Sql]**.

Comme nous l'avons vu au chapitre 2.1., une procédure de rétro-ingénierie nécessite un grand nombre d'informations de types divers. Dans le cas qui nous intéresse (la rétro-ingénierie des bases de données relationnelles) nous trouverons ces informations en différents endroits.

Premièrement, nous pourrons analyser des déclarations faites avec des langages de description et de manipulation de données. Ces langages seront propres à un SGBD. Nous avons choisi de baser notre étude sur le langage SQL car il semble être le plus répandu des langages relationnels.

En suite, les données, elles-mêmes, pourront nous être utiles. En effet, nous pourrons y déceler des empreintes de règles régissant le champ d'application de la base de données.

Enfin, nous pourrons nous baser sur les tables-système du SGBD. Ces tables sont, effectivement, une source d'informations non négligeable. De fait, tout ce qui est géré par le SGBD, y est décrit sous forme tabulaire et accessible via le même langage que les données de l'utilisateur.

Cette quantité de sources entraînera une certaine redondance dans l'information récoltée. Mais, du fait que toutes ces informations sont recueillies sur base d'une même représentation formelle du monde réel, il ne saurait y avoir de conflits entre ces différentes sources. Tout au plus, une règle extraite de l'une sera plus stricte qu'une autre. Nous nous permettons de rappeler qu'il s'agit là d'une caractéristique particulière de la fusion de schémas dans le cadre d'un processus de rétro-ingénierie (cfr. chapitre 2.1.3.2.2.).

Finalement, il est à noter que chaque recherche basée sur une source d'information sera qualifiée d'une part par un coût ou investissement et d'autre part par le niveau de certitude, la pertinence et le niveau de précision (ou niveau de restriction) de l'information récoltée.

Il faudra donc se définir une stratégie d'utilisation des recherches d'information. Ce choix devra se baser d'une part sur les recherches qui sont à notre disposition dans l'atelier logiciel et d'autre part, sur le rapport coût/bénéfice de ces recherches. En général, nous choisirons de nous baser en premier lieu sur une recherche de coût faible, le fruit de celle-ci, même s'il est de moindre qualité, permettra de guider des recherches de coût plus élevé. Cette guidance permettra d'améliorer leur rapport coût/bénéfice.

3.2.1. LE LANGAGE DE DESCRIPTION DES DONNÉES

Ce langage, qui est contenu dans le langage SQL au même titre que le langage de manipulation des données, peut nous donner des informations de trois niveaux : physique, logique et externe (vue). Cette découpe est à mettre en rapport avec l'architecture de système de base de données ANSI/SPARC. Notons que les descriptions propres à ces trois niveaux se retrouvent dans un seul et même langage. Des descriptions physiques et logiques pourront donc se trouver au même endroit et ne seront pas distinctement séparées comme on aurait pu l'espérer.

3.2.1.1. LES DESCRIPTIONS DE STRUCTURES PHYSIQUES

Les descriptions de structures de ce niveau sont à mettre en rapport avec le niveau interne de l'architecture ANSI/SPARC. Ce niveau est le plus proche des spécifications du stockage des données en mémoire secondaire. Il détermine la façon dont on stocke les données sur ces supports physiques.

Dans la plupart des SGBD relationnels nous n'aurons de l'information que sur la répartition des tables dans différents fichiers de base et sur la création d'index. L'organisation interne de ces fichiers de base sera gérée par le SGBD et ne sera que peu paramètrée par le responsable de la base de données.

3.2.1.1.1. LES FICHIERS

Comme nous le voyons à la figure III.1., lors de la déclaration d'une table ou d'un index, on doit spécifier dans quel fichier (espace disque) on veut stocker les informations. L'entièreté de la base de données sera donc répartie sur plusieurs fichiers.

```
CREATE TABLE vendeur        ( nu       NUMBER   (5,0)
                              nom      CHAR     (30)
                              région   CHAR     (20)
                            )

TABLESPACE        di_0

CREATE INDEX      vendeur_I  ON   vendeur  (num

TABLESPACE        di_I
```

Figure III.1.

Le choix d'un fichier pour une structure peut se baser sur plusieurs stratégies :

D'abord, on peut avoir choisi une stratégie se basant sur la sémantique des tables. Dans ce cas, on aura tendance à rassembler dans un même fichier des tables qui ont un lien sémantique entre elles.

Ensuite, un autre type de stratégie est de se baser sur l'optimisation des accès à l'information. Effectivement, le fait que deux fichiers d'une même base de données puissent être sur deux disques différents, peut pousser le concepteur à choisir de répartir dans deux fichiers des structures auxquelles on accède simultanément. Ces le cas d'une table de base et d'un index qui lui est attaché ou de deux tables de base qui font souvent l'objet d'une jointure.

Enfin, ce choix peut être guidé par une stratégie orientée sur la taille des différents fichiers et de leur gestion. Dans le cas où certaines tables sont de taille importante (cas des tables contenant des historiques par exemple) il sera préférable de les placer dans un fichier qui leur sont propre plutôt que de surcharger un fichier que l'on manipule régulièrement.

3.2.1.1.2. LES INDEX

Un index est un fichier qui est utilisé par un processus d'accès aux données selon un ordre établi par la valeur d'une clé.

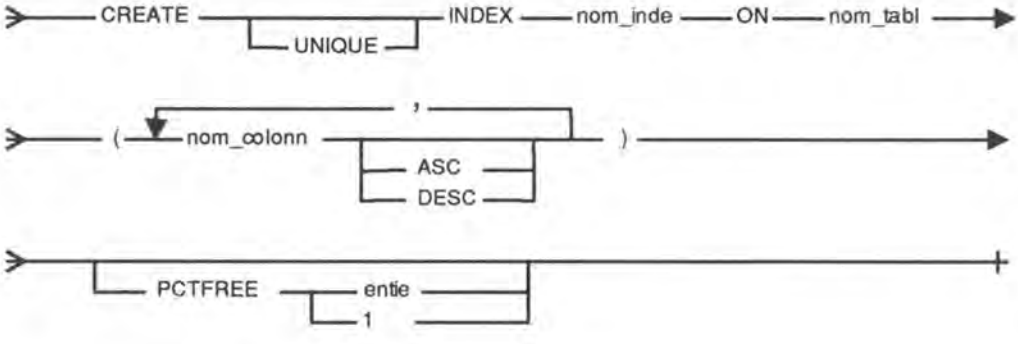


Figure III.2.

Comme nous le voyons dans la figure III.2, qui reprend la syntaxe de création d'un index, cette création peut être paramètrée.

La clause "UNIQUE" indiquera qu'il ne peut exister deux lignes de la table qui possèdent la même valeur pour la clé.

La clause "PCTFREE", quant à elle, nous renseignera sur le pourcentage de l'espace mémoire occupé par l'index que le concepteur désire maintenir libre pour des ajouts ultérieurs. Si ce pourcentage est élevé (il excédera rarement 50%) on pourra en déduire que la table, sur laquelle l'index est défini, sera censée voir sa cardinalité augmenter. Tandis que si le pourcentage est faible, la taille de la table devrait être assez stable.

3.2.1.2. LES DESCRIPTIONS DE STRUCTURES LOGIQUES

Les descriptions de structures logiques sont à mettre en relation avec le niveau conceptuel de l'architecture ANSI/SPARC.

Il est à noter que nous nous intéressons aux structures logiques au sens large. Nous ne nous bornerons pas à l'analyse des concepts de base du modèle relationnel. Mais nous prendrons également en compte les "triggers" et les "checks", car nous les considérons comme des objets à part entière de la base de données malgré le fait qu'ils font appel au langage de manipulation des données.

Les concepts de base, ainsi que les "checks", nous permettront de relever les contraintes statiques régissant la base de données. Ces contraintes correspondent à un prédicat sur l'état courant de la base. D'un autre coté, les "triggers" nous dévoilerons les contraintes dynamiques, c'est à dire les règles portant sur le passage de la base de données d'un état à un autre.

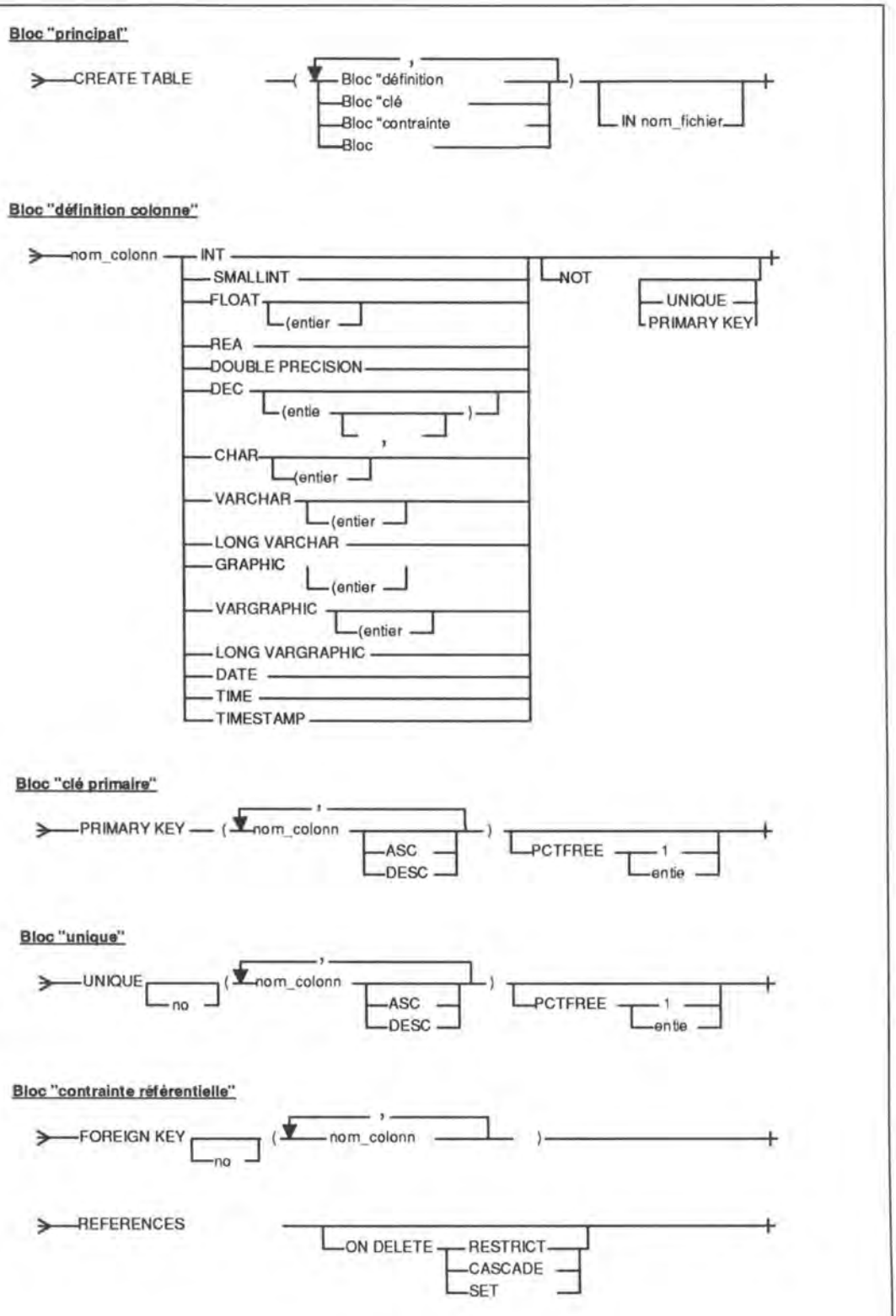

Bloc "principal"
CREATE TABLE
Bloc "définition
Bloc "clé
Bloc "contrainte
Bloc
IN nom_fichier
Bloc "définition colonne"
nom_colonn
INT
SMALLINT
FLOAT
(entier
REA
DOUBLE PRECISION
DEC
(entie
CHAR
(entier
VARCHAR
(entier
LONG VARCHAR
GRAPHIC
(entier
VARGRAPHIC
(entier
LONG VARGRAPHIC
DATE
TIME
TIMESTAMP
NOT
UNIQUE
PRIMARY KEY
Bloc "clé primaire"
PRIMARY KEY
nom_colonn
ASC
DESC
PCTFREE
1
entie
Bloc "unique"
UNIQUE
no
nom_colonn
ASC
DESC
PCTFREE
1
entie
Bloc "contrainte référentielle"
FOREIGN KEY
no
nom_colonn
REFERENCES
ON DELETE
RESTRICT
CASCADE
SET

Figure III.3.

3.2.1.2.1. LES TABLES ET LEURS COLONNES

Comme dit ci-dessus, cette source d'information va nous apporter une description de la base de données en termes de concepts de base du modèle relationnel.

Nous nous proposons, ici, de reprendre ces différents concepts et de voir où ils sont repris dans une définition syntaxique de création de table reprise à la figure III.3.

LES DOMAINES.

Les différents domaines, sur lesquels le SGBD pris en compte permet de définir des attributs, sont mentionnés dans le bloc "définition des colonnes". Le but de cet exposé n'est pas d'exprimer les règles qui régissent ces différents domaines car ils ne sont pris qu'à titre d'exemple. Mais nous noterons quatre remarques essentielles :

Primo : Chaque domaine est clairement défini. En réalité, il s'agit d'une définition en extension que l'on retrouve habituellement dans les manuels d'utilisation des différents SGBD, ainsi que dans leurs tables-système. Ces définitions variant d'un produit à l'autre, nous ne nous attarderons y pas.

Secundo : Il est loisible à l'utilisateur de définir lui-même un domaine et de l'ajouter à la liste des domaines de la base de données.

Tertio : A chaque domaine, une valeur par défaut pourra être associée.

Quarto : Nous noterons que la clause "NOT NULL" permet de retirer du domaine choisi la valeur "NULL".

LES ATTRIBUTS.

Les attributs seront, évidement, définis dans le bloc "définition des colonnes". Leur déclaration précédera la spécification du domaine sur lequel ils sont définis.

LES RELATIONS.

La déclaration d'une relation est reprise dans le bloc "principal". Nous remarquons que l'on pourra lui assigner des attributs, une clé primaire et une clé étrangère par l'intermédiaire des déclarations reprises dans les différents blocs.

LES CLÉS PRIMAIRES.

Une clé peut être déclarée de quatre manières différentes :

- Dans le bloc "définition de colonnes" via la clause "PRIMARY KEY".
- Dans le bloc "clé primaire".
- Dans le bloc "définition de colonnes" via la clause "UNIQUE".
- Dans le bloc "unique".

Les deux premières déclarations serviront à déclarer une clé primaire et donc permettront de déclarer des clés étrangères s'y référant, alors que les deux dernières seront dédiées à la déclaration de clés candidates. Une règle indique qu'une seule clé primaire peut être déclarée sur une table et cela d'une seule manière. Notons que ces quatre déclarations ne sont pas équivalentes. Les déclarations via des clauses dans le bloc "définition de colonne" seront réservées aux clés définies sur un seul attribut.

LES CLÉS ÉTRANGÈRES.

Les clés étrangères seront déclarées dans le bloc "contrainte référentielle". Nous pouvons remarquer que seul le nom de la table référencée est nécessaire. Il n'y a pas besoin de spécifier le nom des colonnes formant la clé primaire. En effet, le SGBD saura automatiquement qu'on se réfère à l'ensemble des colonnes définies comme clé primaire.

La clause "ON DELETE" définit la règle à suivre quand une ligne référée est retirée de la table parent. Ces règles sont au nombre de trois :

- "RESTRICT" : Avant on doit enlever de la table dépendante toutes les lignes qui font référence à cette ligne de la table parent.

- "CASCADE" : L'effacement de la ligne référée entraîne automatiquement la suppression des lignes lui faisant référence dans la table dépendante. Attention, un seul niveau de "cascade" est permis. Donc, une ligne qui suit cette règle ne peut être, en même temps, référencée et référençante.

- "SET NULL" : La clé étrangère de la table dépendante sera automatiquement mise à la valeur "NULL" lors de l'opération de suppression de la ligne référée. Pour pouvoir suivre cette règle, un des attributs sur lesquels est définie la clé étrangère devra accepter la valeur "NULL".

Remarquons que le support de la contrainte de référence n'est que partiel. En effet, rien n'empêche d'insérer une ligne, dans la table dépendante, contenant une clé étrangère dont la valeur ne se retrouverait pas dans l'ensemble des valeurs pris par la clé primaire de la table parent.

3.2.1.2.2. Les triggers

Un trigger (détente, gâchette ou déclencheur en français) est un objet qui détermine au SGBD une action à réaliser avant, ou après, l'exécution d'une requête d'un certain type (création, suppression ou mise à jour). Cette action revêtira elle-même la forme d'une requête. Un trigger représentera, donc, l'expression d'une contrainte posée sur la base de données.

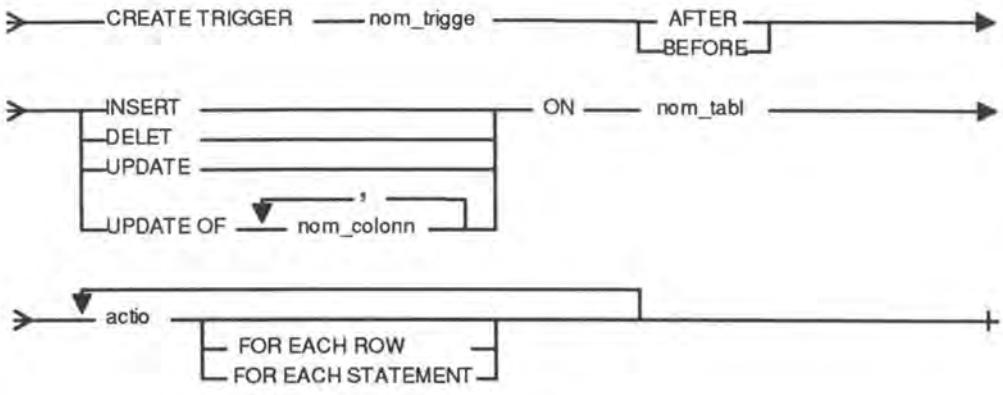


Figure III.4.

Dans la figure ci-dessus, nous trouvons les clauses suivantes :

La clause "BEFORE" indique que le SGBD doit réaliser l'action avant l'exécution de la requête.

La clause "AFTER" indique que le SGBD doit réaliser l'action après l'exécution de la requête.

Les clauses "INSERT", "DELETE", "UPDATE" et "UPDATE OF" indique le type de requête où le trigger déclenchera l'action à réaliser. La clause "UPDATE OF" est un cas particulier de la clause "UPDATE" restreint à certaines colonnes.

La clause "action" est une requête exprimant l'action que le SGBD devra réaliser. Il pourra s'agir d'une insertion, d'une suppression ou d'une modification dans une table.

Enfin, la clause "FOR EACH ROW" indique que l'action est évaluée pour toutes les lignes affectées par la définition du trigger, tandis que, pour la clause "FOR EACH STATEMENT", l'action n'est évaluée qu'une seule fois. Dans ce dernier cas, des références à des valeurs de ligne ne sont pas autorisées dans la déclaration de l'action.

Comme nous le voyons, un trigger, pour réaliser une action, fait appel au langage de manipulation des données. Mais, il n'en demeure pas moins un objet à part entière du SGBD. C'est pourquoi, il trouve sa place dans un chapitre consacré au langage de description des données. De ce fait, nous vous renvoyons au chapitre consacré à ce langage pour consulter les informations supplémentaires que l'on peut retirer de l'analyse des triggers.

Nous pouvons donc conclure que l'analyse des triggers nous apportera bien des renseignements sur les règles qui régissent le domaine d'application de la base de données.

3.2.1.2.3. LES CHECKS

Nous allons découvrir maintenant comment un check peut exprimer une contrainte statique. Nous rappelons, qu'une contrainte statique doit être respectée par toute extension d'une base de données. Elle ne concernera donc pas l'instant de passage, de ces extensions, d'un état à un autre. Signalons que ces contraintes pourront aussi bien être intra-relation qu'inter-relation.

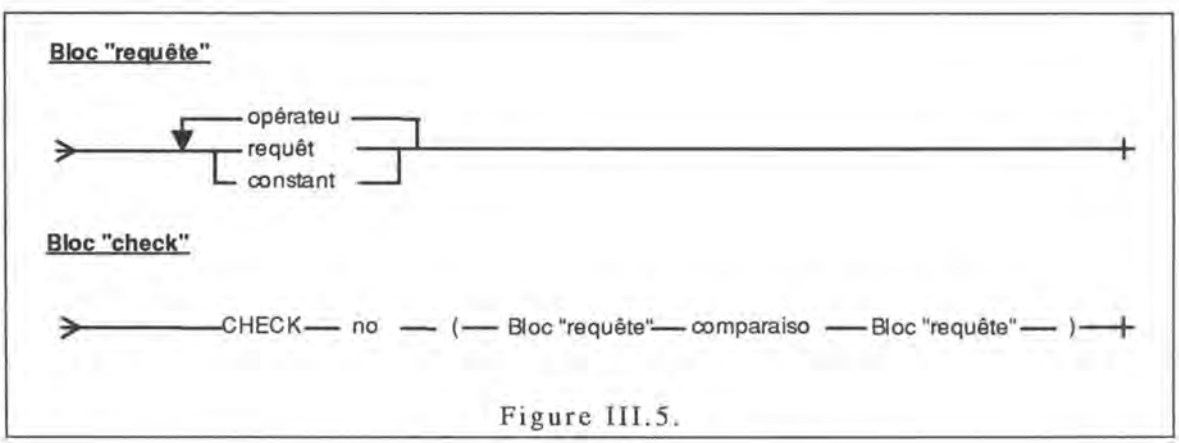


Figure III.5.

La figure III.5. définit d'abord un bloc "requête" comme étant un ensemble non vide de requêtes et/ou constantes liées par des opérateurs qui pourront être de quatre types :

- Les opérateurs de comparaison : $<, \leq, =, \geq, >, \neq$.
- Les connecteurs logiques : AND, OR et NOT.
- Les opérateurs ensemblistes : IN et BETWEEN.
- L'opérateur de comparaison de chaînes de caractères : LIKE.

Mais il est quand même à observer que d'un produit à l'autre l'étendue des possibilités est variable.

La clause "comparaison" se verra assigner, exclusivement, un opérateur de comparaison approprié. Cet opérateur sera le centre de l'expression de la contrainte.

3.2.1.3. LES DESCRIPTIONS DE VUES

Les descriptions des vues sont à mettre en rapport avec le niveau externe de l'architecture ANSI/SPARC. Il s'agit du niveau le plus proche de l'utilisateur et donc du domaine d'application.

Le type d'information que l'on pourra tirer de cette source nous permettra donc de raffiner la vision que l'on a de la base de données. Assurément, vu que ce niveau est proche de l'utilisateur, les données apparaîtront sous la forme la plus compréhensive. Effectivement, la redéclaration de colonnes, que nous observons à la figure III.6, peut nous faire penser que les descriptions des tables ne sont pas en parfait accord avec les objets qu'elles représentent. Par contre, la description d'une vue, elle, doit impérativement correspondre avec l'image de l'objet que l'utilisateur a.

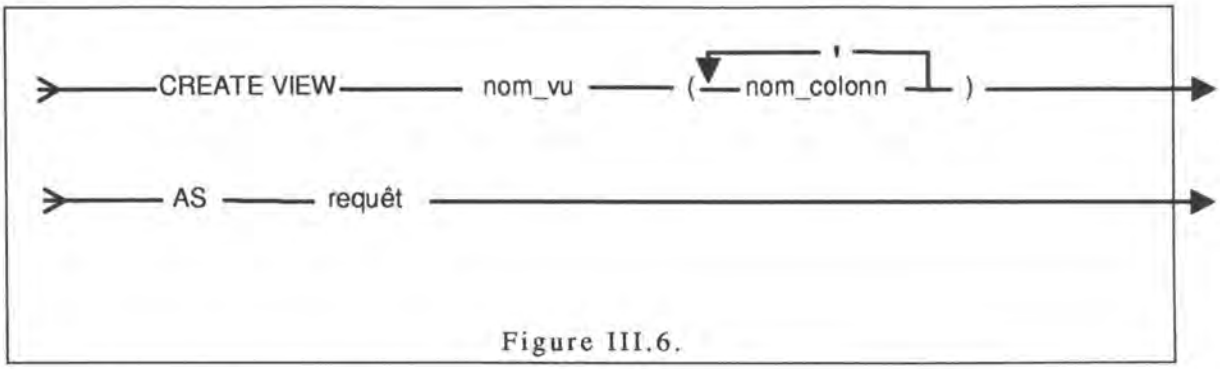


Figure III.6.

Comme nous le remarquons à la figure III.6, la déclaration d'une vue, via la clause "requête", fait également appel au langage de manipulation des données. De ce fait, nous pourrons également recueillir des informations en provenance de ce langage, en particulier les opérations sur les chaînes de caractères et les opérations arithmétiques sur les attributs de type numérique.

3.2.2. LE LANGAGE DE MANIPULATION DES DONNÉES

Le langage de manipulation des données sur lequel nous avons basé notre étude est inclus dans le langage SQL. D'autres types de langage existent mais celui-ci semble le plus répandu à tel point qu'il est devenu un standard.

Le SQL est un langage assertionnel ou non-procédural. Notre but ici n'est pas d'expliquer la syntaxe de toutes les commandes disponibles mais simplement de déterminer le genre d'information que l'on peut en tirer par une brève analyse. Le "survol" de cette source d'information pourra peut-être susciter de l'étonnement. Mais nous rappelons que ce document limite son étude à un sous-ensemble de la rétro-ingénierie qui se focalise sur l'étude des données utilisées par une application (cfr. chapitre 1.1.1. Définition). C'est pour cela aussi que nous nous limiterons aux requêtes exprimées dans les vues, triggers et checks.

Le seul type d'information que nous nous bornerons à analyser ici, concerne l'expression de liens qu'il peut exister entre des tables ou entre des attributs de la base de données.

3.2.3. LE CONTENU DES TABLES

Le contenu des tables est une source d'information que l'on ne prend pas toujours en compte lorsque l'on veut rechercher et comprendre le schéma d'une base de données. Pourtant, on peut y relever les empreintes de règles statiques qui régissent le domaine d'application. Tout comme l'exemple peut aider à comprendre la théorie, l'extension de la base de données peut aider à comprendre celle-ci.

Dans une stratégie de rétro-ingénierie, on ne commencera pas par des recherches basées sur le contenu de tables. Effectivement, un tel type de recherche est consommateur de beaucoup de ressources. Cela est dû à la taille de l'information à traiter.

Par contre, l'affinage de la vision que l'on a du schéma de la base de données est un domaine où cette source d'information nous sera d'une grande utilité. Manifestement, une partie des règles que doit suivre une base de données n'est supportée que par les applications accédant à celle-ci. De ce fait, on ne trouvera trace de ces règles que dans les données sur lesquelles elles portent.

Enfin, il ne faudra jamais perdre de vue que les informations retirées de l'analyse de cette source d'information ne pourront jamais être certaines. Car, n'étant qu'une extension de la base de données, le contenu de celle-ci ne pourra en aucun cas générer une règle générale. Elle ne pourra qu'infirmer une hypothèse.

3.2.4. LES TABLES-SYSTÈME

Comme le prescrit la quatrième règle définie par Codd, un SGBD relationnel doit, pour être qualifié comme tel, supporter un catalogue dans lequel la description de la base de données est représentée au niveau logique sous forme de tables. Ces tables porte le nom de "table-système". Elles sont automatiquement générées lors de la création d'une base de données et entièrement gérées par le système.

L'entièreté de ces tables ne nous sera pas utile dans notre démarche de rétro-ingénierie. C'est pourquoi nous nous contenterons d'en reprendre les principales. Le nom, la structure et l'organisation de ces tables variant d'un produit à l'autre, nous ne donnerons, ci-dessous, qu'une description simplifiée basée sur l'exemple du SGBD Sybase **[Syb]**.

SYSTYPES.

Cette table contient une ligne pour chaque domaine soit fourni par le système, soit défini par l'utilisateur. Parmi les différents attributs de cette table, on en trouvera un qui :

- identifie le domaine.
- indique si le domaine accepte la valeur "NULL".
- fait référence à la procédure qui génère la valeur par défaut.
- fait référence à la procédure qui vérifie les contraintes d'intégrité posées sur ce domaine.

SYSCOLUMNS.

Cette table contient une ligne pour chaque colonne définie dans une table ou une vue. Parmi les différents attributs de cette table, on en trouvera un qui :

- identifie la colonne.
- fait référence à la table ou la vue où elle est définie.
- indique si le domaine, sur lequel est définie la colonne, accepte la valeur "NULL".
- fait référence à la procédure qui génère la valeur par défaut.
- fait référence à la procédure qui vérifie les contraintes d'intégrités posées sur ce domaine.
- la position que la colonne occupe dans sa table.

Comme nous le remarquons, il existe beaucoup de redondance d'information entre cette table et la table "systypes". Cette technique a, apparemment, été préférée à celle de l'attribut de référence.

SYSOBJECTS.

Cette table contient une ligne pour chaque table, vue, check et trigger déclaré. Parmi les différents attributs de cette table, on en trouvera un qui :

- identifie l'objet.
- détermine le type de l'objet.
- contient le nom de l'objet.
- fait référence aux triggers, que ce soit pour l'insertion, la suppression ou la mise à jour.

SYSKEYS.

Cette table contient une ligne pour chaque clé primaire ou étrangère. Parmi les différents attributs de cette table, on en trouvera un qui :

- identifie la clé.
- détermine le type de la clé.
- contient la référence à la table sur laquelle la clé est définie.

De plus, bien sur, il y aura un attribut pour stocker la référence de chaque colonne constituant la clé, le nombre maximum de colonnes dépendant du produit.

Remarquons que certains produits ne gèrent pas cette table. Dans ce cas, c'est à l'utilisateur de le faire.

SYSINDEXES.

Cette table contient une ligne pour chaque index. Dans cette table, en plus des attributs contenant des informations techniques, on en trouvera un qui :

- donne le nom de l'index.
- identifie la table sur laquelle l'index est défini.
- détermine le type d'index.

SYSCOMMENTS.

Cette table contient une ligne pour chaque vue, check, trigger et procédure donnant une valeur par défaut d'un domaine. De plus, l'utilisateur pourra y placer un commentaire sur un autre type d'objet. Dans cette table, en plus des attributs contenant des informations techniques, on en trouvera un qui :

- identifie l'objet auquel s'applique ce commentaire.
- reprend un commentaire si la ligne a été introduite par l'utilisateur ou le texte de la déclaration de l'objet dans le cas contraire.

Notons que le deuxième attribut a une longueur maximum. Au cas où le commentaire (ou la déclaration) serait plus long, il serait repris sur plusieurs lignes différenciées par un numéro de ligne.

3.3. EXTRACTION DES STRUCTURES DE DONNÉES

Comme nous l'avons vu au chapitre 2.1., consacré à la méthodologie de la rétro-ingénierie, l'extraction des structures de données a pour objet de produire une expression du schéma de la base de données en termes de concepts propres au modèle sur lequel elle est construite.

Pour le cas présent, il s'agira donc d'exprimer le schéma d'une base de données relationnelle sous forme de domaines, d'attributs, de tables, de clés primaires et de clés étrangères.

Mais il nous a semblé bon d'apporter trois modifications à cette liste.

Primo : Aux cinq concepts de base du modèle relationnel, il nous a paru utile d'ajouter celui de dépendance fonctionnelle. Il sera l'expression de règles propres à toutes extensions du schéma de la base de données. Ces règles nous permettront de retrouver les identifiants des différentes relations. De plus, elles pourront être utilisées, dans les phases suivantes du processus de rétro-ingénierie, comme bases à des recherches.

Secundo : Nous ne limiterons pas nos recherches aux seules clés primaires. Mais nous nous intéresserons aux clés candidates en général. Ce choix se base sur la simple raison qu'il peut y avoir d'autres liens entre les relations que ceux qui sont explicités. Effectivement, si la définition d'une clé primaire sur un ensemble d'attributs permet de référencer cet ensemble dans la définition d'une clé étrangère, rien n'empêche le concepteur de pouvoir se servir d'autres ensembles d'attributs pour effectuer une jointure entre deux relations.

Tertio : Il nous a également paru bon d'étoffer le concept de domaine. En plus du type sur lequel une colonne est déclarée, nous rechercherons le domaine réel des valeurs contenues dans cette colonne. Nous attacherons également au concept de domaine les notions de valeur minimum et valeur maximum et valeur par défaut.

Après avoir exprimé les tenants et les aboutissants de cette phase d'extraction, décrivons les étapes de celle-ci. Pour illustrer ces étapes, nous nous baserons sur la figure III.7. (reprise du chapitre 2.1.).

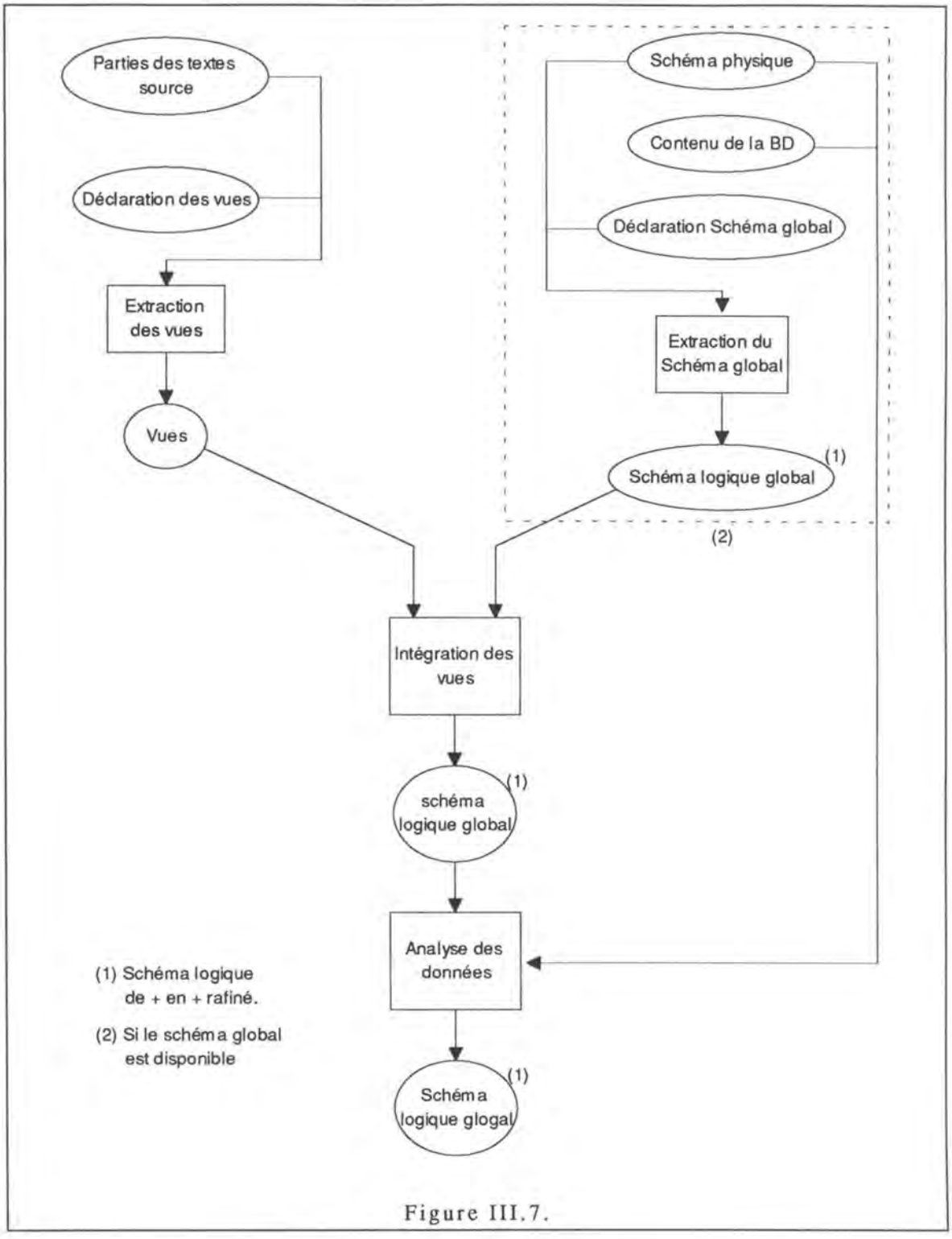
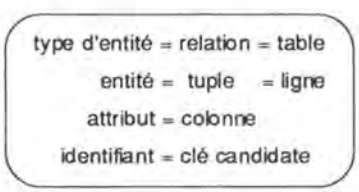

Parties des textes source
Déclaration des vues
Extraction des vues
Vues
Schéma physique
Contenu de la BD
Déclaration Schéma global
Extraction du Schéma global
Schéma logique global
(1)
(2)
Intégration des vues
schéma logique global
(1)
Analyse des données
Schéma logique glogal
(1)
(1) Schéma logique de + en + rafiné.
(2) Si le schéma global est disponible

Figure III.7.

Remarquons tout d'abord que nous sommes dans une situation où le schéma global de la base de données est disponible. Donc, dans ce cas, la phase d'extraction des structures de données se compose d'un processus d'extraction du schéma globale, d'un processus d'intégration des vues et d'un processus d'analyse des données. Cela dit, dans l'application de cette phase, décrite dans ce présent chapitre, nous nous proposons de réaliser ces trois processus en parallèle. Nous ne retrouverons donc pas dans la découpe de ce chapitre la chronologie qui pourrait ressortir de la lecture de la figure ci-avant.

Les résultats des différents processus de la phase d'extraction des structures de données seront représentés dans un modèle de représentation des données unique pour tout le processus de rétro-ingénierie. Il s'agira du modèle entité-association augmenté des concepts propres au modèle relationnel en tenant compte des trois aménagements que nous avons décrits plus haut. De l'utilisation de ce modèle unique, découlent certaines équivalences de termes que nous reprenons dans la figure ci-dessous.



Enfin, une remarque importante à souligner est le fait que l'expression du schéma de la base de données, résultant de cette phase-ci, ne suivra pas la contrainte de relation. En d'autres termes, les relations reprises dans le schéma de la base de données ne seront pas forcément identifiables par l'ensemble de leurs attributs. Ceci sera à prendre en compte dans ce qui suit.

3.3.1. RECHERCHE DES TABLES ET DE LEURS ATTRIBUTS

Pour mener à bien cette recherche, nous proposons d'exploiter les deux sources d'information que sont les tables systèmes et les déclarations de créations de tables. Ces deux sources contenant la même information, il ne sera pas nécessaire de les exploiter de concert.

En ce qui concerne la recherche sur base de déclarations de créations de tables, le processus à suivre sera fort simple. A chaque table, on associera une relation portant le même nom. Cette relation se verra associer un attribut pour chaque attribut de la table. Ici aussi, il y aura équivalence des noms.

Par contre, pour ce qui est de la recherche basée sur les tables systèmes, elle se fera en deux étapes.

La première étape aura pour objet de rechercher dans la table système "sysobject" les lignes qui définissent une table de base. Pour chacune de ces lignes, une relation sera créée. Les noms de la table de base et de la relation seront identiques.

La deuxième étape, elle, s'occupera de trouver les attributs des relations. Pour ce faire, elle recherchera dans la table système "syscolumns" les différentes colonnes associées à une table. A chaque colonne trouvée, on ajoutera un attribut de même nom à la relation adéquate.

Nous avons présenté les deux étapes comme s'exécutant l'une à la suite de l'autre. Mais rien n'empêche d'effectuer la recherche d'une relation et de ses attributs en parallèle.

Remarque.

Lors de notre analyse de bases de données réelles, il nous est apparu que le SGBD ne servait pas qu'à gérer la base de données. En réalité, nous avons remarqué que des tables, que nous pouvons qualifier de "tables parasites", ont été introduites dans la base de données. Nous nous proposons, ici, de décrire l'utilité de ces tables et les indications qui nous ont permis de les détecter.

Le premier type de table parasite a été découvert sur une table nommée "SRW_FIELD". Cette table sert à conserver des paramètres de champs d'acquisition ou de présentation de données à l'écran. Nous avons pu la détecter d'abord grâce à son nom. Le terme "field" ("champ" en français) ne décrivait pas un objet du domaine d'application de la base de données. De plus, "SRW_" préfixait de nombreuses tables suspectes. Enfin, les noms des colonnes de cette table ont attiré notre attention : "SOURCE_QWERY", "TARGET_POSITION", "FORMAT_MASK", "LINES_BEFORE", "SPACE_BEFORE", "RELATIVE_POSITION", "MULTI_PANEL", ...

Ensuite, une table semblable et nommée "SRW_REPPORT" contenait des informations sur le format d'impression de rapport. Nous avons pu la détecter non seulement par son nom, mais aussi par le nom de certains de ses attributs tels que: "PAGE_HEIGHT", PAGE_WIDTH", "LEFT_MARGIN", "RIGHT_MARGIN", ...

Un autre type de table parasite consiste en des tables qui contiennent, dans un attribut de type caractère et assez long, soit du code interprété par le programme, soit une requête SQL pour certains traitements.

Enfin, des tables contenant des historiques ont été découvertes grâce à leur nom et au fait qu'elles soient définies dans un espace disque qui leur était réservé.

A ces tables qui ne font pas partie réellement de la base de données, nous pouvons ajouter celles qui ont servi aux tests et qui portent des noms tels que "TABESSAIS" ou "TABBIDON".

Nous recommandons pour un processus de rétro-ingénierie d'écarter ces tables qui ne sauraient qu'apporter du bruit dans la réflexion engagée lors de ce processus.

3.3.2. Recherche du domaine des attributs

Comme nous l'avons dit au début de ce chapitre, il nous a semblé bon d'étoffer le concept de domaine. En plus du domaine sur lequel est déclarée une colonne (nous l'appellerons à partir de maintenant "le type"), nous avons voulu attacher au concept de domaine une valeur par défaut, une valeur maximum, une valeur minimum et un domaine réel qui est le fruit d'analyses effectuées sur les données. Nous allons, maintenant, décrire la manière de rechercher les composantes de ce concept élargi de domaine.

3.3.2.1. Recherche des types

Nous ne nous attarderons pas sur les types fournis par les différents produits, que nous appellerons types de base. Cette recherche se borne à lire les déclarations de créations de tables et à voir sur quel type est défini chaque attribut.

Par contre, les types définis par les utilisateurs nous paraissent devoir faire l'objet d'analyses plus fouillées car ces types ne sont pas forcément repris dans une documentation.

Pour définir ces types, nous avons à notre disposition la table système "systypes". Nous pouvons y trouver pour chaque type défini par l'utilisateur :

- Un type de stockage sur les supports secondaires, c'est-à-dire un type de base sur lequel se fonde le type défini par l'utilisateur.
- Une longueur de stockage. Cette longueur devra évidement être un multiple de la longueur d'un élément du type de stockage mentionné avant.
- Enfin, une référence à une procédure qui pourra définir des restrictions du type défini par rapport au type de base sur lequel il se fonde.

3.3.2.2. RECHERCHE DES VALEURS PAR DÉFAUT

Nous allons voir, ici, comment découvrir la valeur par défaut d'un attribut.

Dans un premier temps, on pourra analyser la table système "systypes" pour connaître la valeur par défaut des types définis par l'utilisateur. Effectivement, cette table système contient un attribut qui fait référence à une procédure qui génère la valeur par défaut du type décrit par le tuple considéré.

Mais, cette valeur par défaut, s'il y en a une, n'est peut être pas décrite dans les tables système. Ce ne sera, en effet, pas le cas si l'attribut est défini sur un type de base ou si la valeur par défaut est gérée par l'application qui repose sur la base de données.

Donc, il faut être capable de repérer une éventuelle valeur par défaut à partir de l'analyse des données. Pour cela, nous avons dégagé deux heuristiques :

Premièrement, une valeur par défaut peut se différencier d'une autre valeur par sa fréquence dans une table. En effet, on peut supposer qu'approximativement au moins cinq pour cent des tuples contiennent la valeur par défaut pour l'attribut considéré. D'où une analyse des fréquences de chaque valeur, reprise dans une colonne d'une table, s'avère utile.

Deuxièmement, il y a beaucoup de chance que la valeur par défaut se démarque des autres. Par exemple, pour une colonne de type date, on choisira plus facilement "01/01/1900" pour la valeur par défaut que "23/04/1993".

En plus des données, nous pouvons analyser les procédures check. Par exemple, une procédure qui vérifie que la valeur d'un attribut est contenue entre les bornes [0,255] ou est égale à 65536, peut être considérée comme une procédure définissant la valeur 65536 comme valeur par défaut.

3.3.2.4. Recherche de la valeur maximum et de la valeur minimum

Cette recherche servira à limiter le domaine entre deux bornes. Si ces deux valeurs sont déjà définies dans une procédure check, il sera inutile de procéder à une recherche basée sur les données car cette dernière possède un niveau de certitude plus faible.

3.3.2.5. Conclusion

Nous sommes parvenus à retirer des règles pour déterminer le domaine réel d'un attribut. Mais celles qui résultent d'une analyse des données devront être confirmées par l'utilisateur du processus de rétro-ingénierie.

3.3.4. RECHERCHE DES IDENTIFIANTS

Plusieurs sources d'information peuvent être exploitées pour rechercher l'identifiant d'une relation. Certaines de ces recherches se baseront sur des déclarations explicites et auront donc un niveau de certitude maximum. D'autres se baseront sur des hypothèses ou heuristiques et devront, de ce fait, être validées par la personne responsable du processus de rétro-ingénierie.

Rappelons que nous ne nous bornerons pas à rechercher uniquement les clés primaires des relations. Mais que toute clé candidate sera d'un intérêt primordial pour la suite du processus de rétro-ingénierie.

Nous allons reprendre les différentes sources d'information et expliquer comment on peut les exploiter, à quel coût et avec quels résultats.

Les déclarations d'index.

La première source d'information se trouve au niveau physique. Sans conteste, une déclaration de création d'index, si elle mentionne la clause "UNIQUE", nous informe qu'il ne pourra y avoir deux tuples, dans la relation sur laquelle l'index est déclaré, qui possèdent la même valeur pour la clé de cet index. De ce fait, nous considérerons cette clé comme une clé candidate.

Donc, pour rechercher les clés candidates des tables d'un schéma, il faut d'abord sélectionner les déclarations d'index mentionnant la clause "UNIQUE". Ensuite, pour chacune de ces déclarations, il faut extraire le nom de la table de base sur laquelle elle porte et attribuer à la relation correspondante une clé candidate portant sur les attributs mentionnés dans la déclaration.

Cette recherche a un rapport qualité/coût très élevé car on est certain du résultat et la manière de l'obtenir ne consomme pas beaucoup de ressources.

Les déclarations de table.

Comme nous l'avons vu au chapitre précédent, il y a quatre manières, dans une déclaration de création de table, de définir une clé. Ces quatre façons de procéder feront l'objet de deux recherches différentes.

La première recherche porte sur l'analyse des définitions de colonnes. Pour chacune d'entre elles, on recherche la clause "UNIQUE" et la clause "PRIMARY KEY". Chaque fois que cette recherche aboutit, on affecte une clé à la relation adéquate, basée sur l'attribut correspondant à la colonne analysée.

La seconde recherche, elle, s'occupe de rechercher les blocs "clé primaire" et "unique" dans la déclaration de la table. Pour chaque bloc détecté, on relève les noms des colonnes sur lesquelles il est défini. Ensuite, on attribue à la relation adéquate une clé basée sur les attributs dont on a relevé le nom.

Les résultats de ces recherches peuvent être considérés comme certains. De plus, ils seront obtenus à moindre frais en terme de ressources utilisées. Mais nous ne pouvons nous satisfaire des clés qui ont été explicitées. En réalité, une relation pourrait posséder d'autres clés.

De fait, déclarer un ensemble de colonnes comme étant une clé d'une table n'est qu'un moyen d'optimiser les opérations qui porteront sur cette table. En réalité, il s'agira d'entraîner le SGBD à créer un index sur cette table. Donc, si le concepteur n'a pas cru bon d'opérer de la sorte, il n'en demeure pas moins que la relation peut posséder des clés.

Nous allons donc inclure, à l'ensemble des recherches d'identifiant, deux recherches basées sur l'analyse des noms.

La première de ces deux recherches consiste à trouver un lien entre le nom de la relation et le nom d'un attribut. Nous allons analyser maintenant les traces qui pourraient représenter ce lien.

Si un attribut possède le même nom que la relation sur laquelle il est défini, manifestement, il y a beaucoup de chance que ces deux objets de la base de données déterminent le même objet du réel. Ce serait le cas, par exemple, d'une relation "CLIENT" possédant un attribut du même nom. La table décrirait, alors, toutes les caractéristiques propres à un client. Tandis que la colonne attribuerait à un client donné la qualité d'être le client X. Donc, on pourrait en déduire que cette caractéristique identifie un client dans la table. De ce fait, on peut suggérer que cet attribut soit une clé de la relation.

A cette recherche, on pourrait associer une variante où l'attribut ne porterait pas le nom de la relation telle quelle mais une abréviation.

L'autre recherche que l'on pourrait prendre en considération, est le cas où le nom d'un attribut contient le nom de la table (ou une abréviation) préfixé d'un terme tel que "NUM_", "NUMERO_", "IDENT_", "ID_", ...

Observons qu'il vaut mieux se restreindre à quelques préfixes typiques car une technique souvent utilisée pour nommer un attribut est de concaténer le nom de la table avec le nom de la caractéristique déterminée par l'attribut.

En conclusion, nous pouvons affirmer que ces deux techniques ne sont pas consommatrices de grandes ressources (si on se restreint à sélectionner quelques termes) et qu'elles ont une bonne probabilité d'aboutir à la détection d'une clé candidate. Mais, la probabilité de se tromper existe et n'est pas à négliger. De ce fait, les fruits de ces recherches devront soit être confirmés par l'opérateur, soit servir à guider d'autres recherches.

Les déclarations des vues.

Les deux dernières techniques que nous avons mentionnées dans le cadre de l'analyse des déclarations de tables peuvent être également appliquées à l'analyse de déclarations de vue.

L'originalité dans le cas d'une telle analyse se situe dans le fait qu'à une colonne d'une vue ne correspond pas forcément une colonne d'une table. Nous montrons à la figure III.8. les différentes correspondances possibles.

A une colonne d'une vue correspond une colonne d'une

```
CREATE VIEW v_1 (..., id, ...)
      AS SELECT  ....,
                 num_tva,
                 ....
      FROM client
```

A une colonne d'une vue correspond plusieurs colonnes d'une

```
CREATE VIEW v_2 (..., id, ...)
      AS SELECT  ....,
                 CONCAT (organisme, régime)
                 ....
      FROM  entreprise
```

A une colonne d'une vue correspond une partie de colonne d'une

```
CREATE VIEW v_3 (..., id, ...)
      AS SELECT  ....,
                 SUBSTR (date, 1, 4)
                 ....
      FROM assurance
```

Figure III.8.

Dans le cas décrit par la première vue, il ne s'agit que de renommer un attribut. Dans ce cas, rien ne permettait de croire que l'attribut "NUM_TVA" identifiait la table client.

Dans la seconde vue, on peut déduire que l'identifiant d'une entreprise se compose du type de son régime social et d'un nom (ou numéro) d'organisme. Dans cette figure ci, on note donc un ensemble de colonnes qui identifient la table.

La troisième vue, elle, est le cadre d'une technique qui nous permettra plus tard de rechercher des attributs décomposables. Ici, ce n'est pas la date de contraction d'une assurance qui l'identifie mais plutôt le jour et le mois où l'on doit la payer.

Les deux dernières techniques peuvent être mixées. C'est-à-dire qu'à une colonne d'une vue peut correspondre plusieurs parties de colonnes d'une table.

Le langage de manipulation des données.

En préliminaire, nous rappelons que nous retrouverons les requêtes exprimées dans le LMD à trois endroits : Les définitions de vue, les triggers et les checks. Les textes sources des programmes manipulant la base de données pourraient également être pris en compte, mais ils ne seront pas forcément à notre disposition.

Dans le cadre de la recherche d'identifiant, nous nous concentrerons essentiellement sur les jointures effectuées dans les requêtes. Pour mieux comprendre la suite de ce chapitre, regardons la figure III.9. qui reprend le formalisme d'une requête SQL basée sur une jointure

Remarque : X est un attribut de T1, Y est un attribut de T2, une constante ou une requête donnant comme résultat une seule valeur.

Figure III.9.

Rappelons que la jointure des tables R et S suivant les ensembles d'attributs A_R et A_S est une sélection effectuée sur le produit de R et S selon le critère "valeurs (A_R) = valeurs (A_S)". Cette définition, comme nous allons le montrer, se retrouve dans les requêtes SQL. De fait, les relations dont on doit faire le produit sont reprises dans la clause "FROM" tandis que le critère de sélection est exprimé à la clause "WHERE".

C'est dans le critère de sélection que nous allons pouvoir retrouver un identifiant de T_1 et/ou de T_2. Pour cela, retirons de ce critère un ensemble C de contraintes d'égalité entre un attribut de T_1 et un attribut de T_2. De cet ensemble C, nous pouvons extraire l'ensemble A_1 des attributs de T_1 et l'ensemble A_2 des attributs de T_2.

Selon une recherche basée sur le contenu des tables (ou sur les dépendances fonctionnelles), regardons si A_1 (respectivement A_2) est identifiant de la table T_1 (respectivement T_2).

Soulignons le fait que cette recherche ne saurait affirmer avec certitude que l'un de ces ensembles est un identifiant. De ce fait, elle devra être validée par un responsable du processus de rétro-ingénierie.

Mais, par contre, cette recherche pourrait très bien infirmer l'hypothèse que A_1 (respectivement A_2) soit un identifiant de T_1 (respectivement T_2).

A ce stade, nous pouvons nous retrouver dans trois situations différentes : Soit A_1 et A_2 sont identifiant de leur table respective, soit l'un des deux est identifiant, soit ils ne le sont aucun.

La deuxième situation intéressera également la recherche de clés étrangères qui devrait se faire en parallèle avec cette recherche ci.

Les tables système.

Les tables système pouvant être exploitées pour la recherche de clé candidates sont "syskeys" et "sysindexes".

La première contient une ligne pour chaque clé primaire déclarée. Dans chacune des lignes, on trouve la référence des colonnes constituant la clé et d'une table de base. Il ne suffit plus qu'à attribuer, à la relation adéquate, une clé primaire définie sur les colonnes déterminées.

Sur la seconde, on pourra sélectionner les lignes reprenant les caractéristiques d'un index défini avec la clause "UNIQUE". Le reste du traitement sera identique à ce qui a été dit dans le paragraphe consacré à la recherche basée sur les index.

3.3.5. LA RECHERCHE DES CLÉS ÉTRANGÈRES

Nous n'allons pas, dans cette recherche, nous limiter aux clés étrangères explicitées dans la déclaration des tables. Nous envisageons de relever tous les attributs, ou groupes d'attributs, que l'on peut suspecter de faire référence à une autre relation. Si notre hypothèse est confirmée, alors nous mémoriserons les coordonnées de la clé étrangère et de la table référencée.

Dans ce chapitre, nous proposons différentes sources d'information. Pour chacune d'elles, nous reprendrons les indices qui pourraient nous conduire à la clé étrangère.

Mais, à partir du moment où on a trouvé une clé étrangère et fait le lien avec la table qu'elle référencie, une dernière information doit être cherchée : Il faut savoir si le rapport entre l'ensemble des valeurs de la clé étrangère et l'ensemble des valeurs de la clé candidate, est un rapport d'inclusion ou un rapport d'égalité. Pour rechercher cette information, nous proposons trois méthodes de niveau de certitude et de coût croissant. Notons que ces trois méthodes n'ont pas un niveau de certitude suffisant pour pouvoir se permettre de négliger la confirmation de l'opérateur.

La première méthode se base sur les domaines réels dont la recherche a été expliquée au début du chapitre. Elle consiste à comparer le domaine de la clé étrangère et le domaine de la clé candidate pour savoir s'ils sont identiques, l'un inclus dans l'autre ou non compatibles. Cette recherche ne nécessite pas beaucoup de ressource mais ne peut que réfuter une hypothèse. C'est pourquoi, après l'obtention d'un résultat positif, il est préférable d'utiliser la seconde méthode.

Cette seconde méthode consiste à prendre un échantillon de la table dépendante et de vérifier si l'ensemble des valeurs pour la clé étrangère est inclu dans l'ensemble des valeurs de la clé candidate dans la table parent.

Enfin, la dernière méthode est identique à la seconde mais on ne se contente pas d'un échantillon. Cette méthode a un coût très élevé et demande quand même l'approbation de l'opérateur. De ce fait, il est peut être préférable de s'en tenir aux deux premières méthodes.

Les déclarations d'index.

Un index qui n'est pas déclaré avec la clause "UNIQUE" peut quand même être à la base d'une optimisation d'accès aux données via une clé. Il nous vient donc directement à l'esprit que cette clé pourrait être une clé étrangère. Effectivement, lors de la jointure, le SGBD doit accéder à une table via sa clé étrangère.

C'est pourquoi les ensembles d'attributs qui forment des clés d'index, qui sont définis sans la clause unique, seront suspectés d'être des clés étrangères. Les résultats de cette recherche devront faire l'objet d'une confirmation de la part de l'opérateur.

Les déclarations de tables.

Nous proposons d'exploiter cette source d'information de deux manières différentes.

D'abord, nous pouvons nous baser sur le bloc "contrainte référentielle". Ce bloc nous permet de relever avec certitude et précision l'existence d'une clé étrangère. De plus, nous mémoriserons la clé primaire de la table référencée. Cette recherche a un niveau de certitude maximum et un coût relativement faible. Mais, elle ne permet de retrouver que les clés étrangères explicitées dans le SGBD.

Pour dépasser cette restriction, nous pouvons procéder à une recherche basée sur les noms des colonnes. En effet, si certains de ceux-ci incluent le nom d'une autre table, on peut suspecter qu'ils y font référence. Nous prendrons comme exemple l'attribut de la table "COMMANDE" qui porte comme nom "NUM_CLIENT". Cet attribut fait sûrement référence à la table "CLIENT". Cette recherche a un niveau de certitude moindre mais donne quand même de bons résultats car cette technique d'appellation des attributs est fort répandue. Un point négatif est cependant à relever : Les opérations de comparaison de sous-chaînes de caractères augmentent fortement le coût d'une telle recherche.

Les checks

La contrainte de référence, sous-jacente à l'utilisation de clés étrangères, étant une contrainte statique, on pourra retrouver son expression dans des procédures check. Cette procédure aura la forme d'une contrainte d'inclusion telle que représenté ci-après :

```
CREATE TABLE t_name
 (...,
  CONSTRAINT c_name
   CHECK ( ( SELECT fk
             FROM table_1 ) IN ( SELECT id
                                 FROM table_2 ) )
 );
```

Figure III.10.

Le langage de manipulation des données.

Cette recherche se fait en parallèle avec la recherche d'identifiant basée sur la même source. Comme nous l'avons vu à la fin du chapitre consacré à cette recherche, on aboutit à trois cas de figure concernant les deux ensembles d'attributs sur les quels la jointure est basée :

- Les deux ensembles sont identifiant de leur relation.
- L'un des deux ensembles est identifiant de sa relation.
- Aucun des deux ensembles n'est identifiant de sa relation.

Dans le premier cas, la seule opération à effectuer est la mémorisation du fait que ces ensembles d'attributs se font réciproquement référence.

En ce qui concerne le second cas, l'ensemble d'attributs qui n'est pas identifiant peut être considéré avec certitude comme une clé étrangère.

Le dernier cas représente le résultat de la décomposition d'une relation sur base d'une dépendance multivaluée **[Hai89]**. Le seul traitement que nous envisageons est de fusionner les deux relations liées en une seule. Le contenu de celle-ci sera le résulat d'une opération de jointure sur les deux relations initiales.

Les tables système

Deux tables système peuvent être prises en considération : "syskeys" et "sysindexes". La première subira un traitement semblable à celui des blocs "contrainte référentielle" dans la déclaration de table, tandis que, la recherche basée sur la seconde sera proche de celle effectuée sur base des déclarations d'index.

3.3.6. CONCLUSION

Comme nous venons de le voir, les concepts de base du modèle relationnel ont été extrait des différentes sources d'informations en tenant compte des aménagements que nous y avons apportés. Au terme de cette phase nous obtiendrons donc bien un schéma logique relationnel de la base de données.

3.4. CONCEPTUALISATION DES STRUCTURES DE DONNÉES DÉPENDANTE DU MODÈLE RELATIONNEL

Nous allons, dans ce chapitre, nous concentrer sur la traduction du schéma logique en termes de concepts de niveau d'abstraction supérieur. En d'autres mots, nous allons voir comment traduire un schéma relationnel en un schéma Entité-Association simplifié.

Pour ce faire, nous rechercherons dans le schéma logique le résultat de transformations de concepts propres au modèle Entité-Association. Ces concepts qui n'ont pas d'équivalent dans le modèle relationnel, sont au nombre de quatre : Les attributs facultatifs, décomposables, multivalués et les types d'association. Au terme de cette recherche, nous aurons donc une expression du schéma de la base de données qui suit le modèle Entité-Association.

La plupart des transformations utilisées dans ce chapitre proviennent de **[Hai93]**.

3.4.1. RECHERCHE DES ATTRIBUTS FACULTATIFS

Dans le modèle relationnel et contrairement au modèle Entité-Association, les attributs facultatifs ne sont pas admis. Le concepteur d'une base de données relationnelle doit donc transformer un tel type d'attribut en un concept propre au modèle relationnel.

Deux familles de transformations sont à sa disposition. Soit il transforme l'attribut facultatif en un attribut obligatoire, soit il le transforme en une relation.

3.4.1.1. RECHERCHE D'ATTRIBUTS FACULTATIFS REPRÉSENTÉS PAR DES ATTRIBUTS OBLIGATOIRES

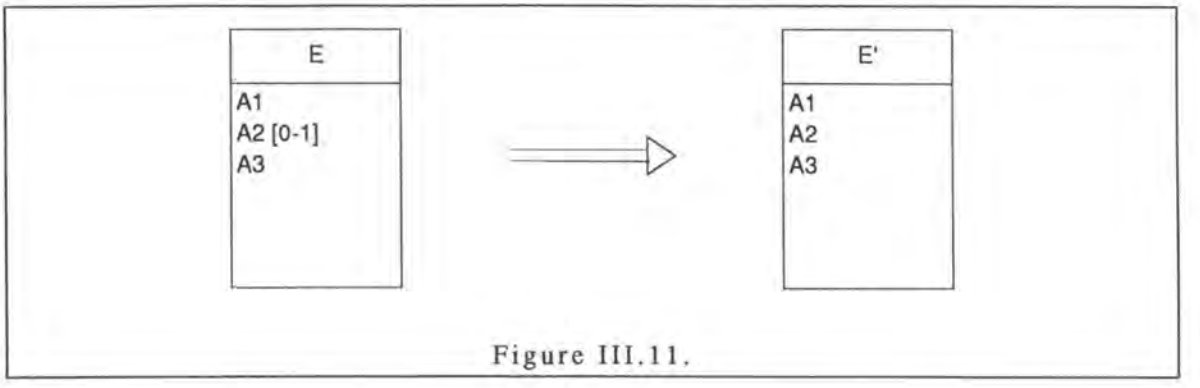


Figure III.11.

La figure III.11. reprend l'aspect général de la transformation qu'a subi l'attribut facultatif. Deux possibilités s'offrent alors au concepteur : Soit E'.A2 est définit sur le même domaine que E.A2, soit sur un domaine différent.

Dans le premier cas, une valeur appartenant au domaine de E.A2 (et de E'.A2) est assignée à l'attribut quand l'information n'est pas connue ou n'est pas applicable. Il va de soit que cette valeur ne pourra jamais être affectée à l'attribut comme information connue et applicable.

Nous retrouverons, par exemple, le cas de la valeur "1/JAN/1900" prise par un attribut dont le domaine est une date

Dans le cadre de la rétro-ingénierie, la recherche d'un attribut ayant subit une telle transformation se base essentiellement sur la définition du domaine précis sur lequel l'attribut a été défini. Lors de l'extraction des structures de données, le domaine de chaque attribut a pu se voir affecter une valeur par défaut. Si celle-ci est comprise dans le domaine alors nous nous trouvons en face d'un attribut facultatif qui a subit la transformation décrite ici.

Dans le second cas, il s'agit essentiellement de l'ajout de la valeur "NULL" au domaine de l'attribut. La recherche de cette transformation est également basée sur le domaine.

Une seule différence intervient entre ces deux recherches : La première devra être confirmée par l'utilisateur tandis que la seconde offre un niveau de certitude tel que la confirmation peut être considérée comme superflue et ce malgré quelque cas pervers.

3.4.1.2. Recherche d'attributs facultatifs représentés par des types d'entité

Comme nous le remarquerons à la vue des figures illustrant les transformations qui nous préoccupent, cette recherche ne peut s'effectuer qu'après avoir explicité les types d'association.

Deux techniques de représentation d'un attribut par un type d'entité sont envisageables. Nous allons les expliquer et étudier la manière dont on peut les retrouver.

3.4.1.2.1. La technique de représentation par instance

Dans la figure III.12 , nous pouvons observer une transformation d'un attribut en un type d'entité. Dans cette transformation, une entité représente une instance de l'attribut. C'est-à-dire, qu'il y aura autant d'entités de type EA2 que d'entités de type E qui possèdent une valeur pour l'attribut A2.

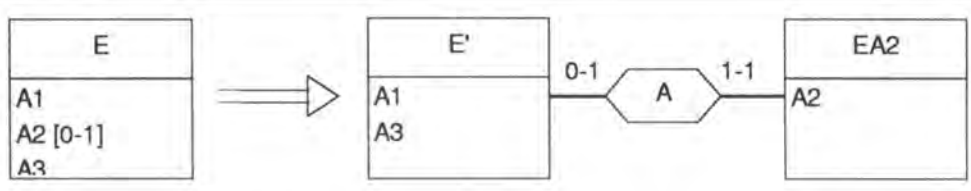


Figure III.12.

Pour détecter le fruit d'une telle transformation, nous nous baserons sur le nombre d'attributs contenus dans les types d'entité et les connectivités posées sur les association où ces types d'entité jouent un rôle. Incontestablement un type d'entité (EA2) en correspondance avec un autre (E) via une association (A) et contenant un seul attribut (A2) peut être suspectée de représenter un attribut du type d'entité avec lequel il est associé (E). De plus, quand le rôle joué par ce dernier (E) dans l'association (A) possède une connectivité 0-1, on peut en déduire que l'attribut représenté par le type d'entité est facultatif. La connectivité 1-1 du rôle joué par le type d'entité EA2 sera la signature de la technique de représentation par instance.

Il nous suffira donc de relever dans le schéma de la base de données un couple de types d'entité associés qui vérifient ces caractéristiques.

3.4.1.2.2. La technique de représentation par valeur

La transformation présentée dans la figure VII.3 est une variante de la précédente. En réalité, dans ce cas ci, chaque entité EA2 ne représente plus une instance de l'attribut A2 de E mais une valeur prise par cette attribut pour une ou plusieurs entités de type E. De ce fait, A2 devient identifiant de EA2 et la connectivité du rôle joué par celui-ci est 1-N.

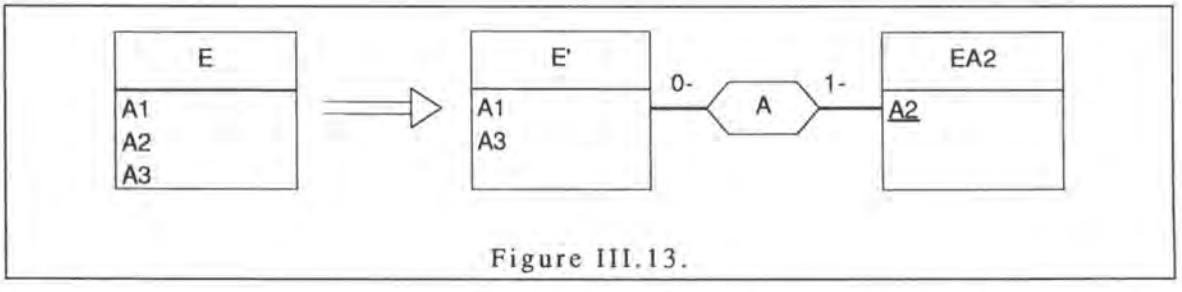


Figure III.13.

La technique de détection de l'utilisation d'une telle transformation est identique à celle utilisée précédemment.

Notons que si E.A2 est identifiant, la connectivité du rôle joué par EA2 sera "1-1". Nous serons donc dans un cas proche de la représentation par instance, à ceci près que EA2 possède un identifiant.

3.4.2. RECHERCHE DES ATTRIBUTS COMPOSÉS

Les attributs composés devront, également, être transformés lors de l'implémentation d'une base de données relationnelle.

Dans ce cadre, nous retrouvons également deux familles de transformations. La première transforme les attributs composés en attributs élémentaires, tandis que la seconde les transforme en types d'entité.

3.4.2.1. RECHERCHE D'ATTRIBUTS COMPOSÉS REPRÉSENTÉS PAR DES ATTRIBUTS SIMPLES

Les techniques de représentation d'attributs composés par des attributs simples sont au nombre de deux. Elles se différencient par le nombre d'attributs simples qu'elles génèrent pour un attribut composé. La première que nous proposons n'en génère qu'un seul tandis que l'autre en produit un pour chaque composant de l'attribut composé.

Nous noterons que la qualité d'identifiant se transmet aux attributs simples générés.

3.4.2.1.1. TRANSFORMATION PAR CONCATÉNATION

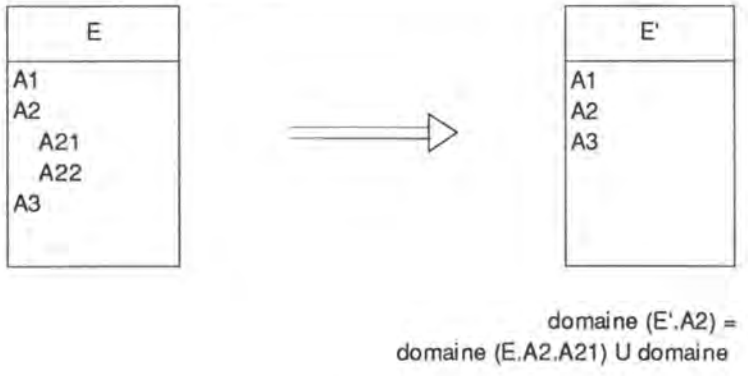


Figure III.14.

La figure III.14. représente la transformation d'un attribut composé en un attribut simple par concaténation. Cella veut dire que l'attribut A2 d'une entité E' contiend la concaténation des valeurs des attributs A2.A21 et A2.A22 d'une entité E.

Avant d'aller plus en détails dans notre recherche, remarquons qu'il y a une perte de sémantique dans cette transformation. Incontestablement, la structure interne de l'attribut E.A2 représentait une organisation des données que l'on ne retrouve pas dans la structure de l'attribut E'.A2.

Dans le cadre d'un processus de rétro-ingénierie, cette perte d'information réduit considérablement le niveau de certitude de nos recherches. De fait on ne saurait passer d'un niveau peu organisé à un niveau bien organisé sans une source nouvelle d'information.

Notons, quand même, quelques indices qui pouraient nous permettre de détecter l'usage de cette transformation.

Premièrement la longueur de l'information contenue dans la colone E.A2 est à prendre en compte. Manifestement, un attribut de type caractère et de longueur 60 pourait attirer notre attention. Le nombre 60 est pris ici à titre d'exemple mais le nombre que l'on déterminera comme étant la limite pour attirer notre attention ne devrait pas tellement s'en écarter.

Deuxièmement, le contenu de ladite colonne peut être aussi le point de départ d'une prise de conscience de l'utilisation de la concaténation. Effectivement cette dernière ne s'applique qu'aux attributs de type caractère. Donc, si E.A2.21 et E.A2.A22 sont de type numérique, il faudra transformer leur contenu en une chaîne de caractères. Cette chaîne de caractères sera de préférence de longueur fixe pour pouvoir, sans dificulté, accéder aux informations. De ce fait, on devra préfixer chaque valeur n'atteignant pas la longueur maximum de "0" ou de " " suivant le type de valeurs représentées. Ceci constitue une empreinte de la transformation non négligeable.

On peut, également, oppérer de la sorte pour les attributs de type date, heure et réel. Par exemple, pour les attributs de type date la recherche de constantes faite à la phase d'extraction des données nous donnerons la chaîne : "??/???/????". Tandis que la recherche de type nous donera :"NNSAAASNNNN" (N = numérique, A = alphabétique et S = caractère spécial).

Enfin, les déclarations de vues sont des sources d'information de première qualité. En effet, si nous reprenons l'exemple de la figure III.14, une vue déclarée sur E' poura définir une colonne sur la première partie de l'attribut A2 et une seconde sur le reste de cet attribut. Comme nous le voyons à la figure III.15, de cette déclaration, on pourra déduire que E'.A2 peut être décomposé en deux parties.

```
CREATE VIEW entité_E (A1, A2_A21, A2_A22, A3) as
                    SELECT
                      A1
                      substr (A2, 1, 5)
                      substr (A2, 6, 5)
                      A3
                    from E'
```

Figure III.15.

3.4.2.1.2. TRANSFORMATIONS PAR DÉCOMPOSITION

La figure III.16. reprend le format général d'une transformation par décomposition d'un attribut composé. Nous observons également ici la perte d'information due à la déstructuration de l'attribut E.A2.

Mais, contrairement à ce que l'on a vu pour la transformation par concaténation, les attributs de E' n'ont pas de caractéristiques pouvant attirer notre attention.

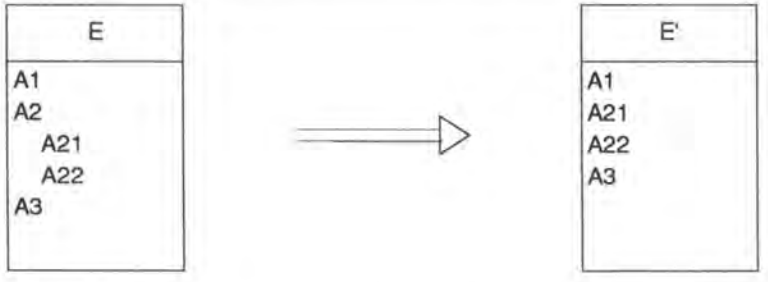


Figure III.16.

La seule condition pour que l'on puisse détecter l'usage de cette transformation est l'utilisation de la technique du préfixe (ou du suffixe). Dans ce cas, le nom de l'attribut E'.A21 sera le fruit de la concatenation du nom de l'attribut E.A2 (ou une abréviation de celui-ci) et du nom de E.A2.A21. Comme nous le voyons sur la figure III.17., dans ce cas, il sera possible de détecter des attributs possédant le même préfixe (ou suffixe) et de les regrouper.

Mais attention ce regroupement ne poura s'effectuer qu'avec l'accord d'une personne connaissant bien la base de données.

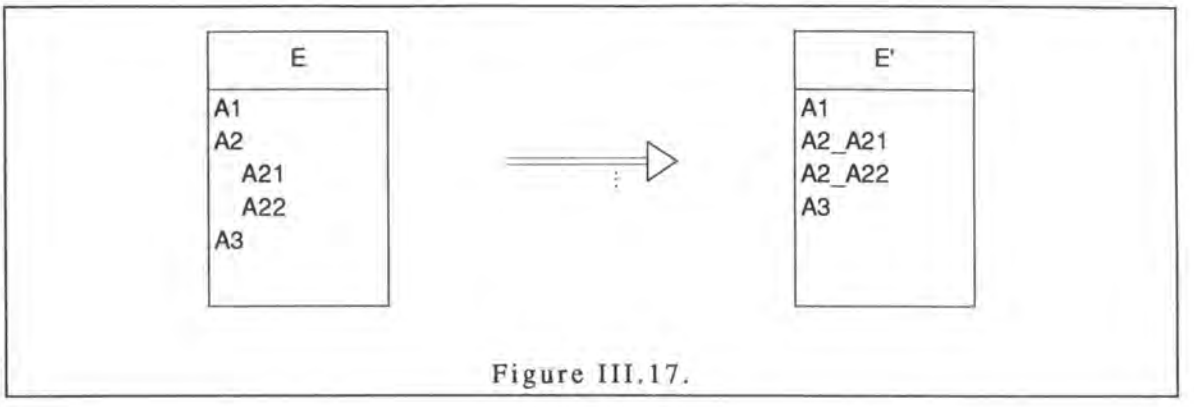


Figure III.17.

3.4.2.2. Recherche d'attributs composés représentés par des types d'entité

Tout comme nous l'avons remarqué lors du chapitre consacré aux attributs facultatifs, cette recherche ne peut s'effectuer qu'après avoir explicité les types d'association.

Deux techniques de représentation d'un attribut par un type d'entité sont, ici aussi, envisageable. En fait, il s'agit de la composition de deux transformations de base : D'une part la représentation d'un attribut simple et obligatoire par un type d'entité et d'autre part la transformation par décomposition d'un attribut composé. Nous allons les expliquer et étudier la manière dont on peut les détecter leur utilisation.

3.4.2.2.1. La technique de représentation par instance

Dans la transformation d'un attribut composé reprise à la figure III.18, pour chaque instance du type d'entité E, une instance du type d'entité EA2 est créée.

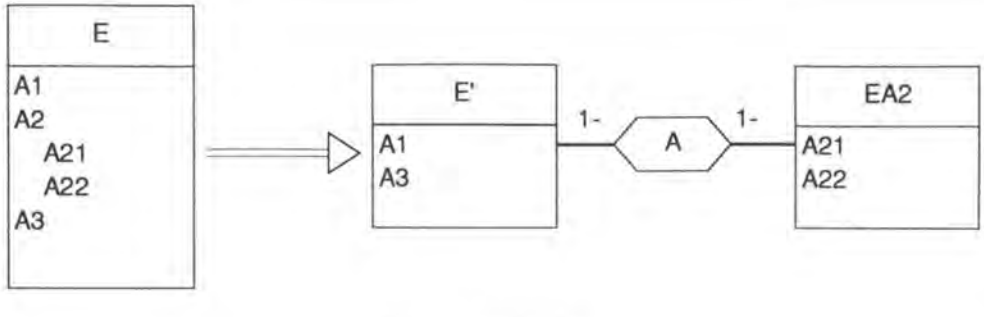


Figure III.18.

Pour pouvoir retrouver le schéma initial, nous nous baserons sur deux remarques : La première remarque à faire porte sur le nombre d'attributs contenus dans le type d'entité EA2. Sans conteste, ce nombre est peut élevé. La seconde remarque est que la connectivité des rôles joués par les types d'entité dans le type d'association A peut permettre de penser que le type d'entité E et le type d'entité EA2 représentent des qualités différentes d'un même objet du réel.

Il nous suffira donc de retrouver les associations posédant des connectivtés 1-1 de vérifier que l'un des deux types d'entité ne possède pas beaucoup d'attributs.

Mais cette recherche n'a pas un haut niveau de certitude. Indubitablement, on ne peut pas concidérer avec certitude que tout type d'association "one to one" représente un attribut multivalué.

En conséquence, nous devrons également nous baser sur les noms des attributs de EA2. Si ceux-ci sont préfixés ou sufixés du nom, ou d'une abréviation du nom, du type d'entité E, alors on pourra admettre avec plus de certitude qu le type d'entité EA2 représente un attribut composé du type d'entité E.

3.4.2.2.2. La technique de représentation par valeur

En ce qui concerne la transformation d'un attribut composé reprise à la figure III.19, pour chaque couple de valeurs pris par les composants de l'attribut A2 du type d'entité E, une instance du type d'entité EA2 est créée.

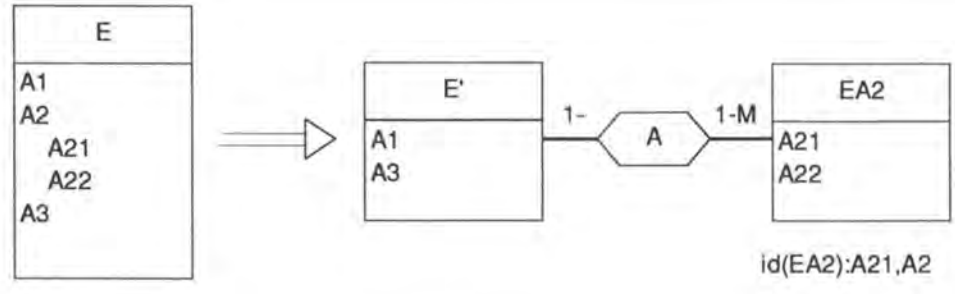


Figure III.19.

Pour rechercher l'utilisation de cette technique, nous utiliserons le même type de recherche que celle décrite ci-avant. La seule différence portera sur la connectivité du rôle joué par EA2 et l'identification de celui-ci. Mais, ici aussi, nous devrons rendre la recherche plus certaine par l'analyse des noms des attributs de EA2.

Notons que si E.A2 est identifiant, la connectivité du rôle joué par EA2 sera "1-1". Nous serons donc dans un cas proche de la représentation par instance, à ceci près que EA2 possède un identifiant.

3.4.3. Recherche des attributs multivalués

Nous allons aborder les différentes techniques de représentation d'un attribut multivalué dans le modèle relationnel. De plus, nous verrons les moyens de déceler, lors d'un processus de rétro-ingénierie, l'utilisation de ces techniques.

La nécessité de ces technique vient du fait que le modèle Entité-Association autorise les attributs multivalués tandis que le modèle relationnel ne le permet pas.

Dans un premier temps, nous nous attarderons au cas où les valeurs de l'attribut multivalué ne possèdent ni doubles, ni notion d'ordre. Ensuite, nous analyserons les différences qu'apportent ces deux principes aux techniques de représentation et aux moyens de les détecter.

3.4.3.1. LES ATTRIBUTS MULTIVALUÉS SANS NOTION D'ORDRE ET SANS DOUBLE

Nous avons relevé cinq techniques de représentation d'un attribut multivalué, compatibles avec le modèle relationnel. Nous allons les présenter et expliquer le moyen de détecter leur utilisation.

3.4.3.1.1. LA TECHNIQUE DE REPRÉSENTATION DES INSTANCES.

Présentation

La technique de représentation des instances est reprise à la figure III.21. Il s'agit de transformer l'attribut répétitif en un type d'entité. A chaque instance de l'attribut A2 corespondra une entité de type EA2, d'où le nom de "représentation des instances". Nous pouvons remarquer que la connectivité du rôle joué par le type d'entité E' est la même que celle liant l'attribut A2 au type d'entité E.

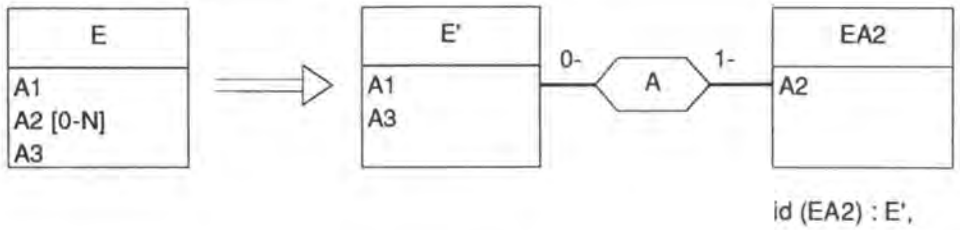


Figure III.21.

Moyens de détection.

La particularité du schéma résultant de cette transformation se situe au niveau du schéma de la base de données. Assurément, il n'est pas habituel de voir un type d'entité comme EA2 qui ne contient qu'un attribut et ne joue un rôle que dans un seul type d'association.

Nous rechercherons donc un type d'entité jouant un rôle de connectivité 1-1 dans un seul type d'association. De plus, ce type d'entité ne devra posséder qu'un seul et unique attribut qu'il soit composé ou non. Enfin, Il sera identifié par le couple formé de son attribut et du rôle qu'il joue dans le type d'association.

A ce stade de description, le niveau de certitude de la recherche est élevé. En réalité, l'équivalence des deux représentations est certaine. Mais l'intention du concepteur de représenter de la sorte un attribut répétitif l'est moins.

Pour augmenter ce niveau de certitude nous pouvons également procéder à une analyse des noms. Si le nom du type d'entité EA2 est la concaténation des noms (ou d'une abréviation des noms) du type d'entité E' et de l'attribut EA2.A2, il sera quasiment certain que EA2 représente un attribut multivalué de E'. Il en ira de même si le nom de l'attribut A2 contient comme préfixe le nom du type d'entité E' ou une abréviation de ce nom.

3.4.3.1.2. La technique de représentation des valeurs.

Présentation

La technique de représentation des valeurs est proche de celle vue au point 3.4.1.1.. Ici, le type d'entité EA2 représente non pas une instance de l'attribut E.A2 mais une valeur que cet attribut prend pour un ou plusieur type d'entité E. De ce fait, il identifie le type d'entité EA2.

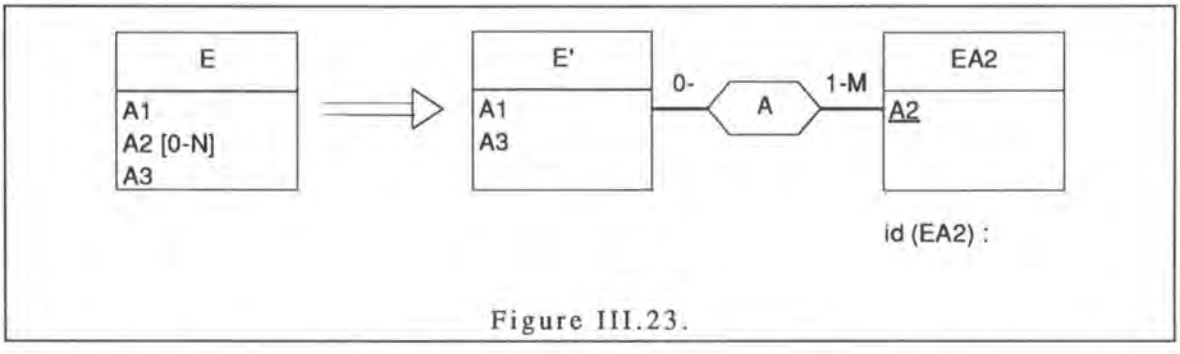


Figure III.23.

Nous remarquerons à la figure III.23. que le type d'association A est "many to many" contrairement au type d'association présenté pour la technique de représentation par instance. De fait, la connectivité du rôle joué par E' est identique à celle de l'attribut E.A2 tandis que la connectivité du rôle joué par EA2 exprime le nombre d'entité de type E où une valeur donnée de A2 peut apparaitre.

Notons que si E.A2 est identifiant, la connectivité du rôle joué par EA2 sera "1-1". Nous serons donc dans un cas proche de la représentation par instance, à ceci près que EA2 possède un identifiant.

Moyens de détection.

Les caractéristiques du schéma résultant de cette transformation se trouvent dans le schéma de la base de données comme c'est le cas pour la technique précédente. Nous rechercherons donc un type d'association binaire de connectivité "many to many" où un des deux types d'entité sur lequel il est défini ne possède qu'un seul attribut qui l'identifie.

Cette recherche possède un niveau de certitude élevé qui peut être encore amélioré par l'analyse des noms expliqué pour la transformation précédante.

3.4.3.1.3. La technique de l'augmentation de l'identifiant (dénormalisation)

Présentation

Cette technique est simple et donc courrament utilisée. Il s'agit d'une variante de la technique de représentation des instances. Effectivement, pour chaque instance de l'attribut E.A2 une entité de type E' sera créée. Celle-ci sera identifiée par le couple formé de la valeur de cet attribut E.A2 et l'identifiant de son entité d'origine.

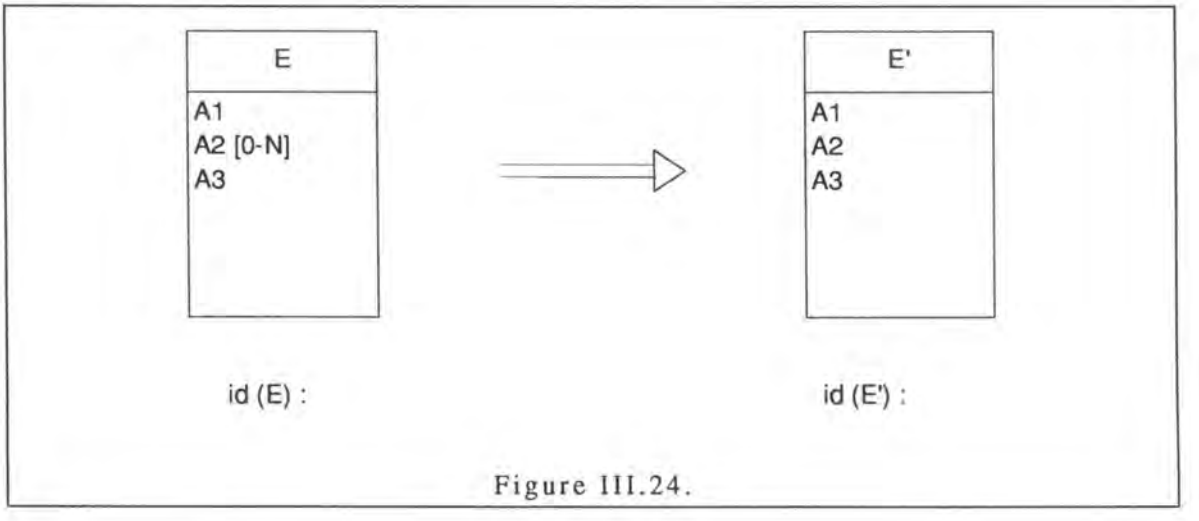


Figure III.24.

Moyens de détection.

Pour détecter l'utilisation de cette technique, on ne saurait pas se baser sur la cardinalité des relations (nombre de tuples) ni sur le schéma de la base de données. On devra plutôt se baser sur des règles déduites de l'extension des relations. La caractéristique sur laquelle on se basera est le fait que E'.A1 est à lui seul identifiant de E' si on ne tient pas compte de l'attribut E'.A2. Cette caractéristique est la trace du type d'entité original E.

Nous chercherons donc un type d'entité dont l'identifiant est composé et contient au moins un attribut. (Il pourait ne contenir que des rôles.) Si lorsqu'on retire du type d'entité un des attributs composant son identifiant ce type d'entité est toujours identifié par ce qui reste de cet identifiant, alors l'attribut retiré était en fait un attribut multivalué. Pour vérifier cette règle, on peut utiliser les dépendances fonctionnelles comme il est illustré à la figure III.26. (où A1 et A3 peuvent être considérés comme des groupes d'attributs).

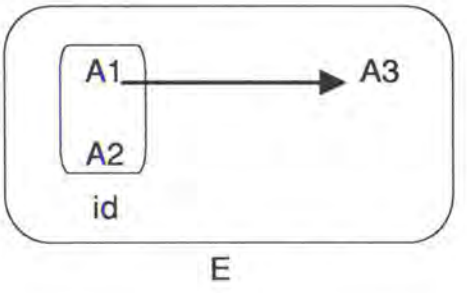


Figure III.26.

La règle peut alors s'énoncer ainsi : "Si l'identifiant d'un type d'entité comporte un attribut qui ne détermine fonctionnellement aucun autre attribut alors on peut déduire que cet attribut peut en fait être représenté par un attribut multivalué."

Remarquons que si E.A2 est identifiant, il devient difficile de repérer l'utilisation de cette transformation. Mais la construction E' n'est-elle pas préférable à la construction E.

3.4.3.1.4. LA TECHNIQUE DE L'INSTANCIATION.

Présentation

Comme nous le constatons dans la figure III.27, la technique de l'instanciation consiste à représenter chaque instance d'un attribut multivalué par un attribut monovalué. Ce dernier sera définit sur le même domaine que le premier.

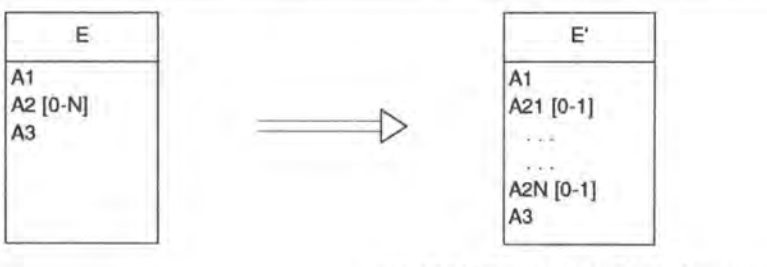


Figure III.27.

Mais nous pouvons avoir une vision plus générale de cette transformation comme nous le voyons dans la figure III.28.

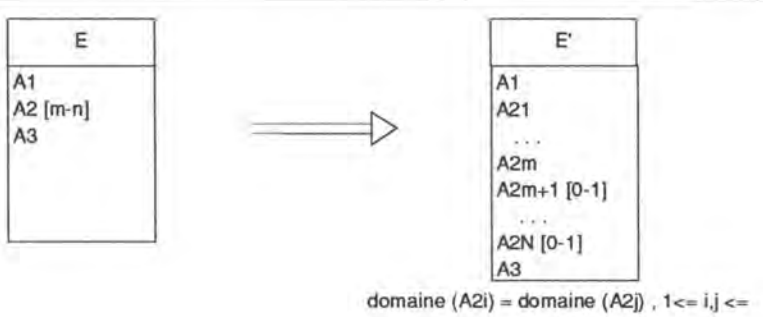


Figure III.28.

Notons également que cette transformation n'est applicable que dans le cas où la borne supérieure N est connue.

Moyens de détection.

Nous remarquerons que la caractéristique émanant de cette technique de représentation se situe au niveau du degré d'une relation (nombre d'attributs).

Nous serons donc à la recherche d'un type d'entité défini sur un grand nombre d'attributs. Nous avons constaté que souvent ceux-ci présentaient quelques particularités :

- Ces attributs se suivent dans la définition du type d'entité.
- Ils sont tous définis sur le même domaine (en exceptant la valeur "NULL").
- Ils peuvent se diviser en deux groupes qui se suivent; le premier est composé d'attributs obligatoires, le second d'attributs facultatifs.
- Leur nom est identique a un suffixe (ou préfixe) près qui est souvant un chiffre allant de 1 à N. De plus ils sont triés dans l'ordre (souvent croissant) de ces chiffres
- Enfin, ces particularités ne se retrouvent pas forcément dans la définition du type d'entité mais peuvent apparaitre dans la déclaration d'une vue définie sur ce type d'entité uniquement.

Le niveau de certitude de cette recherche sera d'autant meilleur que le type d'entité possède plusieurs de ces caractéristiques.

Remarquons que si E.A2 est identifiant, alors tous les attributs générés par cette transformation sont des identifiants de E'.

3.4.3.2. LA TECHNIQUE DE LA CONCATÉNATION

Présentation.

Comme nous le voyons à la figure III.30, cette technique consiste à concaténer la représentation en caractère des différentes instances de l'attribut multivalué. Le nombre maximum d'instance ("N") devra donc être connu.

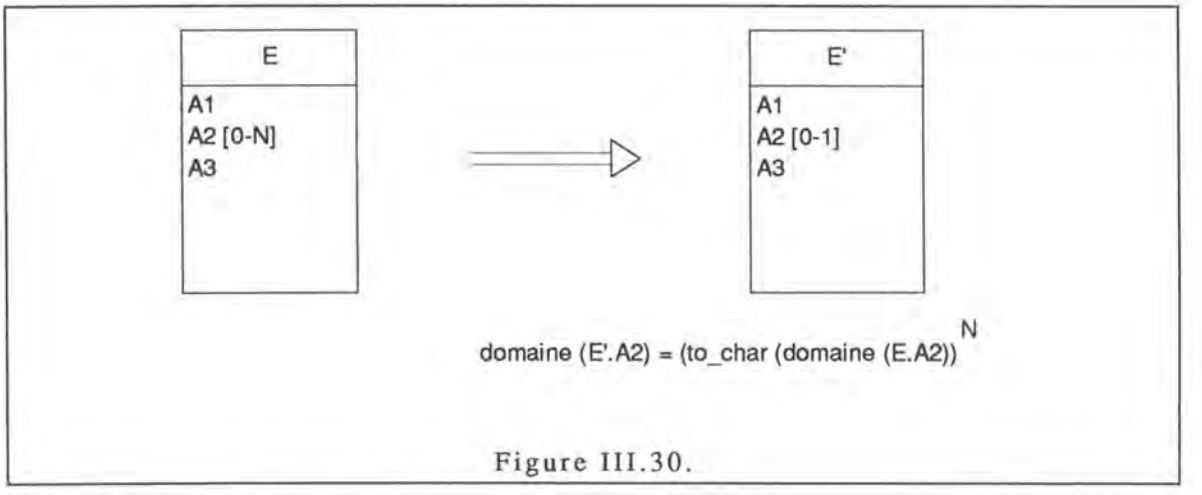


Figure III.30.

Moyens de détection.

Cette technique ne joue ni sur la cardinalité de la relation ni sur son degré.

De fait, les principales caractéristiques découlant de cette transformation se trouvent au niveau du domaine sur lequel est défini l'attribut E'.A2. Effectivement, celui-ci possède quelque particularités :

- Ce domaine est toujours du caractère.
- Il n'est généralement pas bien approprié. C'est-à-dire que le domaine réel retrouvé lors de l'extraction des données n'est pas en concordance avec le type caractère. Il s'agira de nombres, de dates,...

- De plus, dans ce domaine réel, un cycle peut intervenir. Il s'agit soit d'un caratère de séparation définit par l'utilisateur, soit d'un caractère propre au domaine réel qui revient a interval régulier ("/" pour les dates, "." pour les réel...)
- Enfin, on retrouve des caractères de remplissage soit pour compléter les valeurs qui n'ont pas une longueur standard (un chiffre trop petit sera précédé de "0"), soit pour remplir la place non occupée quand il y a moins de N instances représentées.
- Une autre source d'information peut être exploitée : Les vues. L'une d'elle pourait être définie sur au moins N colonnes, chacune garnies avec une sous-chaîne de l'attribut E'.A2.

3.4.3.3. Cas où il peut y avoir des doubles dans les valeurs prises par l'attribut multivalué

Pour les trois premières techniques que nous avons vues au chapitre précédent, le fait qu'il y ait des doubles dans l'attribut multivalué entraîne une perte d'identifiant. Cette perte peut être compensée par une technique appelée "technique de condensation". Il s'agira en fait d'ajouter un compteur à la construction ayant perdu son identifiant.

Nous allons donc reprendre ces différentes techniques, revoir la manière de détecter leur utilisation, leur appliquer la technique de condensation et voir comment on peut repérer son utilisation.

Par contre, pour les techniques d'instanciation et de concaténation, la situation est inchangée au niveau du schéma. La différence est au niveau des valeurs contenues dans les attributs de remplacement.

3.4.3.3.1. La technique de representation des instances.

Les changements dûs à l'existance de doubles

La seul différence avec ce qui a été dit au point 3.4.1.1. est le fait, comme le montre la figure III.32, que le type d'entité EA2 n'a plus d'identifiant.

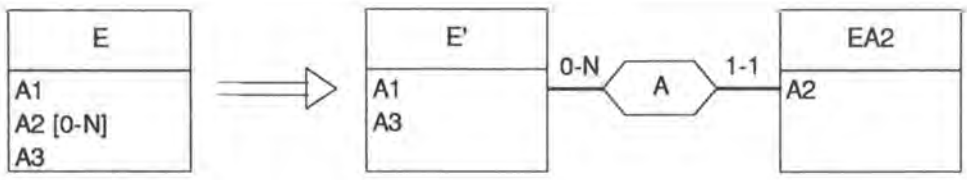


Figure III.32.

La manière de détecter l'utilisation de cette technique restera donc fort semblable.

La technique de condensation.

La perte de l'identifiant du type d'entité EA2 est due au fait que deux entités de ce type ayant la même valeur pour A2 puissent être associées à une même entité de type E'. Il suffit donc de représenter toutes les instances d'un attribut A2 d'une entité qui ont la même valeur par une seule entité de type EA2 qui posséderait un attribut, "count" par exemple, reprenant leur nombre.

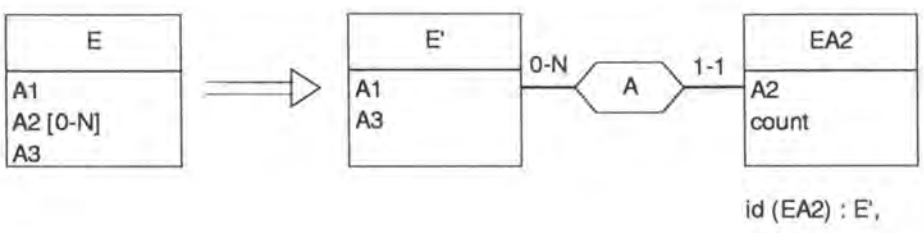


Figure III.33.

Par cette technique, comme nous le voyons à la figure III.33, le type d'entité EA2 retrouve son identifiant.

Moyens de détection.

La recherche de l'utilisation de cette technique sera la même que celle expliquée au point 3.4.1.1. La seul différence qui se marquera est la presence d'un deuxième attribut dans le type d'entité EA2. Ce nouveau venu sera surement de type numérique et de petite taille (possibilité de représenter des valeurs de 0 à 255).

La certitude de cette recherche est d'un bon niveau. Ce niveau sera rehaussé si le deuxième attribut se nomme ou contient dans son nom un terme tel que "count", "compt", "num", ...

3.4.3.3.2. La technique de representation des valeurs.

Les changements dûs à l'existence de doubles.

Avec la technique présentée au point 3.4.1.2., on ne peut représenter des attributs multivalués contenant des valeurs en double. Cela vient du fait qu'une association est identifiée par les rôles assumés par les entités qu'elle met en relation. Donc, un couple d'entités ne peut être mis en association plus d'une fois.

Mais cette restriction ne s'applique qu'aux types d'association. En conséquence, le concepteur qui veut représenter par la technique de représentation des valeurs un attribut multivalué contenant des valeurs en double, sera tenté de remplacer ce type d'association en un type d'entité. Cette transformation aura pour résultat le schéma repris à la figure III.34.

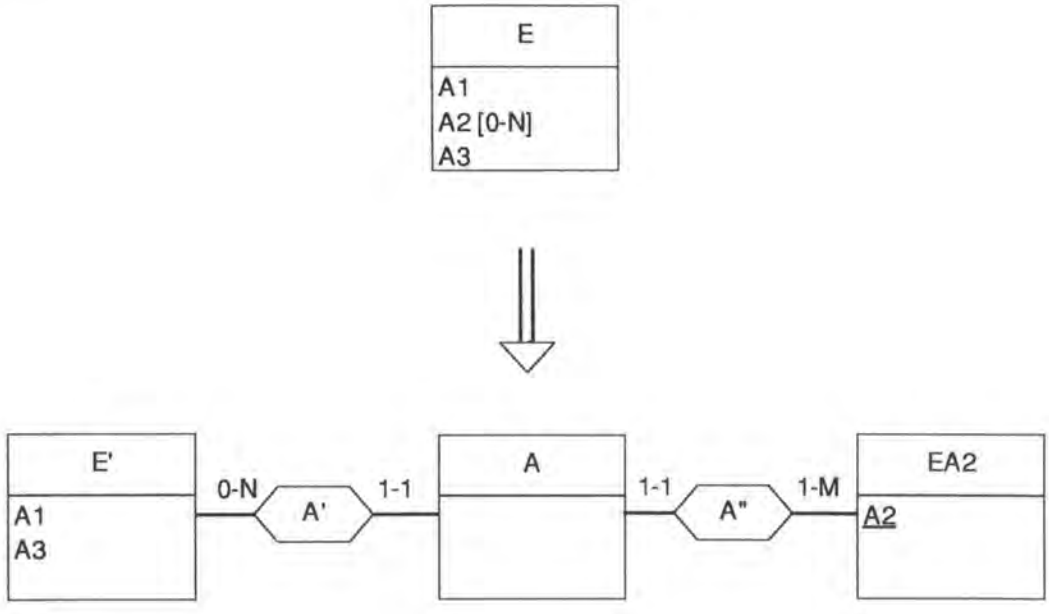


Figure VII.24.

Figure III.34.

Nous remarquons que le type d'entité A remplace le type d'association de même nom dans la figure VII.13. et qu'il ne possède pas d'identifiant.

La technique de condensation.

Ici aussi, la technique de condensation peut être employée. Comme nous le voyons à la figure VII.25, il suffit d'ajouter un attribut numérique au type d'entité A. Cette attribut reprend le nombre de fois que l'attribut E.A2 recueille la valeur reprise par l'attribut A2 de l'entité de type EA2 qui lui est liée.

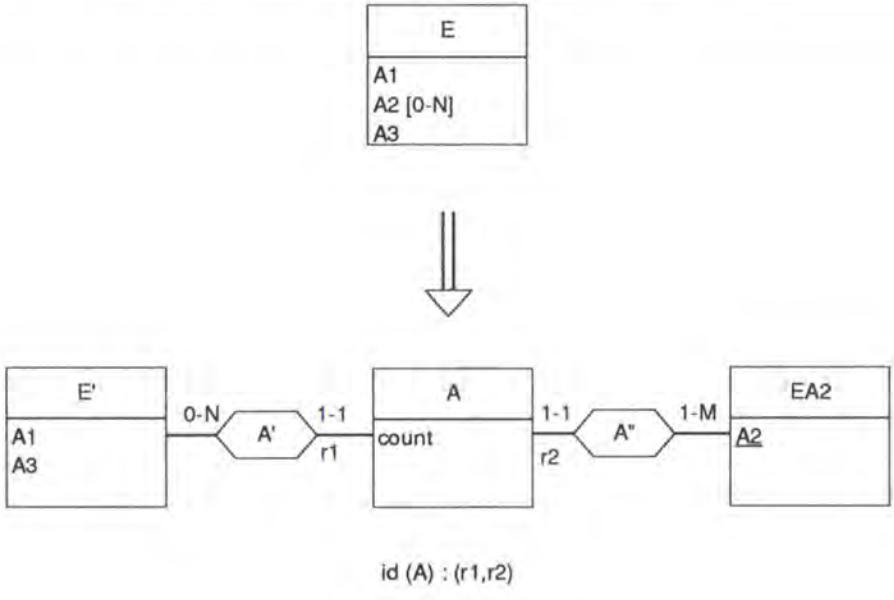


Figure III.35.

Moyens de détection.

Cette construction, qu'elle utilise la technique de condensation ou non, est assez typique. D'où une technique d'investigation qui ne sera pas compliquée : On recherchera un type d'entité n'ayant pas d'attribut et jouant un rôle, de connectivité 1-1, uniquement dans deux type d'association "one to many". De plus un de ces types d'association le liera à un type d'entité ne comptant qu'un attribut, identifiant de surcroît.

Si le type d'entité recherché possède un attribut numérique de petite taille (représentant des valeur de 0 à 255), alors il y a beaucoup de chance que l'attribut représenté par la construction décrite ci-dessus accepte des valeurs en double. De plus, dans ce cas et uniquement dans ce cas, le type d'entité recherché possédera un identifiant. Celui-ci sera composé des deux seuls rôles qu'il joue.

3.4.3.3.3. La technique de l'augmentation de l'identifiant (dénormalisation)

Les changements dus à l'existance de doubles.

Ici aussi, nous remarquerons une perte de d'identification due à l'existance de valeurs doubles.

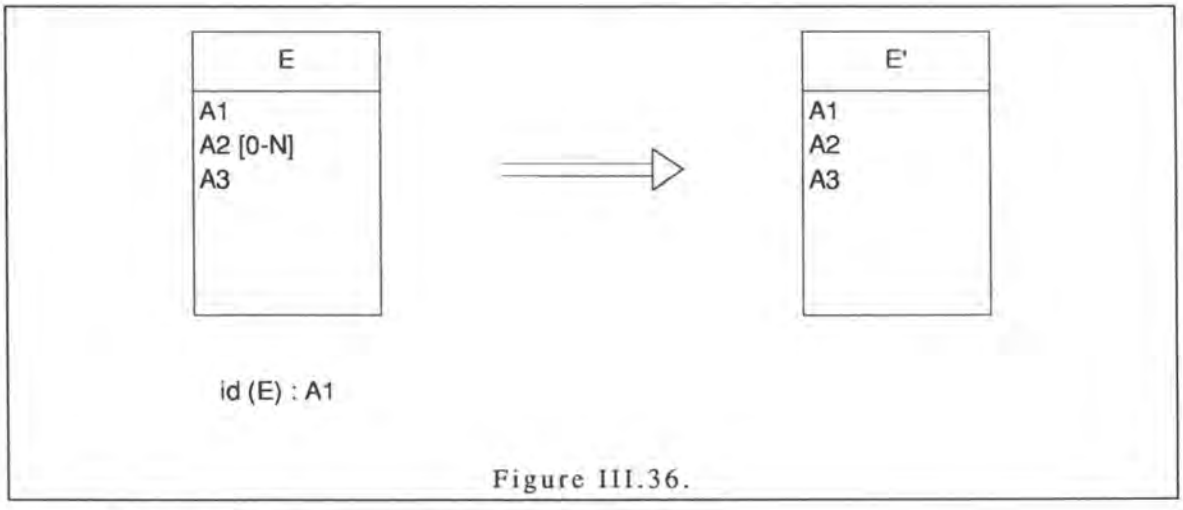


Figure III.36.

La technique de condensation.

Tout comme pour les techniques de représentation précédentes, l'identifiant du type d'entité EA2 pourra être retrouvé grâce à la technique de condensation.

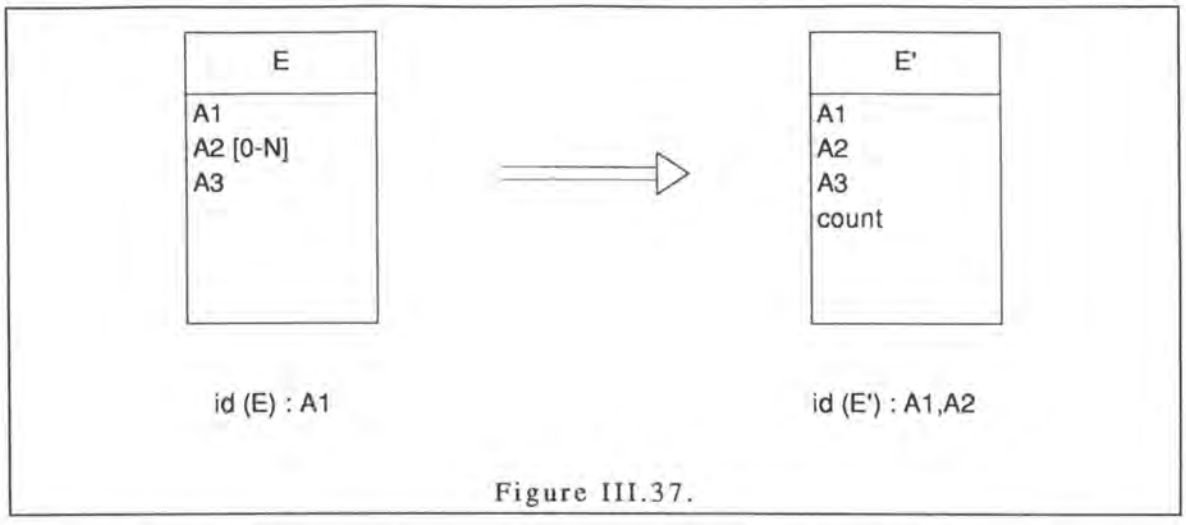


Figure III.37.

L'attribut supplémentaire reprendra le nombre d'entités possédant les mêmes valeurs pour le couple (A1,A2).

Moyens de détection.

La recherche basée sur les dépendances fonctionnelles reste d'utilité dans ce cas de figure. Mais, si on n'a pas utilisé la technique de condensation, l'identifiant déduit de l'analyse de ces dépendances fonctionnelles n'aura put être exprimé dans la base de données pour cause de lignes doubles dans la table représentant le type d'entité EA2. Ce cas est décrit à la figure ci-dessous.

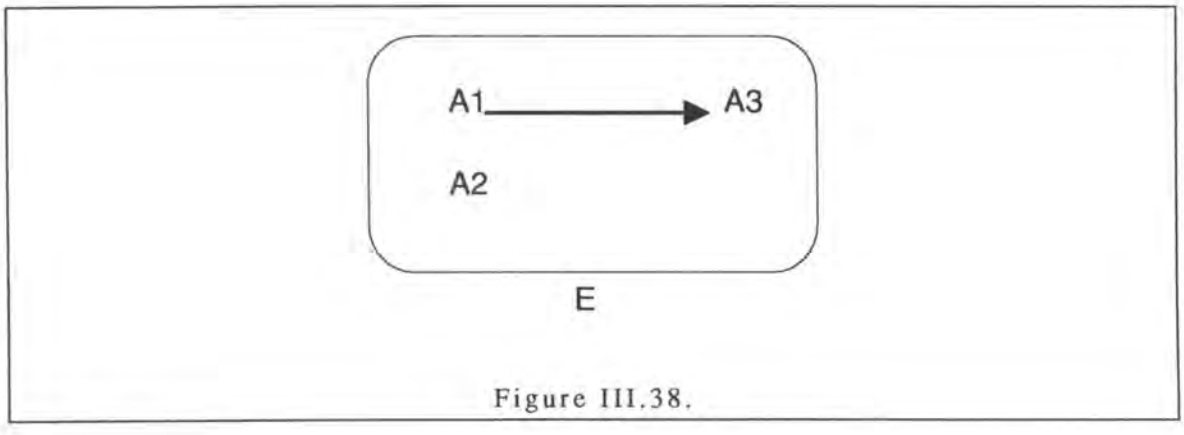


Figure III.38.

De plus, si on a utilisé la technique de condensation, alors il faudra s'assurer de l'existance, dans le groupe d'attributs A3, d'un attribut numérique dont le nom est ou comporte "count", "compt", "num", ...

3.4.3.3.4. La technique de l'instanciation et la technique de la concatenation

Vu que l'attribut multivalué est remplacé par un ou plusieurs autres attributs dans le même type d'entité, le seul changement apporté par la possibilité qu'il contienne des valeurs en double se trouve au niveau des valeurs contenue par ce ou ces attributs de remplacement. Effectivement :

- Soit des attributs de remplacement par la technique de l'instanciation auront la même valeur.
- Soit ont retrouvera plusieurs fois la même valeur dans les sous-chaînes de l'attribut de remplacement par concaténation.

3.4.3.4. Cas où l'on doit prendre en compte une notion d'ordre

Nous pouvons envisager deux types d'ordonnancement des valeurs des attributs multivalués.

Dans le premier type, l'ordonancement est dépendant de la valeur des instances de l'attribut. En d'autres mots, Les instances de l'attribut multivalué sont classées dans l'ordre croissant ou décroissant.

Dans le second type, l'ordonancement est indépendant de la valeur des instances de l'attribut. Le classement est donc fait selon un critère non représenté dans la base de données. De ce fait on ne saurait le déceler qu'à condition qu'une technique d'indexage explicite aie été utilisée.

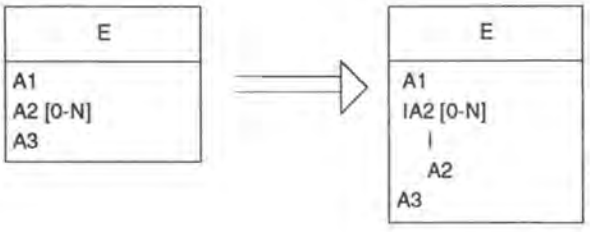


Figure III.39.

Une technique d'indexage explicite, comme nous le voyons à la figure III.39, consiste à associer à chaque valeur de l'attribut multivalué un numéro d'ordre. Ceci nous oblige à définir l'attribut comme étant composé de deux valeurs : L'une reprenant la véritable information et l'autre le numéro d'ordre. Une fois cette transformation effectuée, il s'agira d'un cas d'ordre dépendant de la valeur du numéro ajouté.

Maintenant que nous avons vu le moyen de réduire tous les cas à la notion d'ordre dépendant de la valeur, nous allons pour chaque technique de représentation d'un attribut multivalué voir les façons d'implémenter cet ordre puis comment on peut les détecter.

Nous remarquerons que le fait que l'attribut multivalué soit un identifiant n'apporte aucune modification aux raisonnements.

3.4.3.4.1. Les techniques de représentation des instances et de représentation des valeurs

Pour ces deux techniques il est assez difficile de déceler une notion d'ordre. La représentation de l'attribut multivalué est souvant accédée par une jointure ce qui fait qu'une recherche sur les index n'est pas perspicace.

La seule place où on pourait relever une trace est une vue reprenant pour chaque ligne une entité et une des valeurs de son attribut multivalué. Si dans la déclaration de cette vue la commande de tri "ORDER BY" est utilisée sur la valeur de l'attribut multivalué alors on peut déceler l'importance de l'ordre.

3.4.3.4.2. La technique de l'augmentation de l'identifiant (dénormalisation)

La figure III.40. reprend les trois situations possibles.

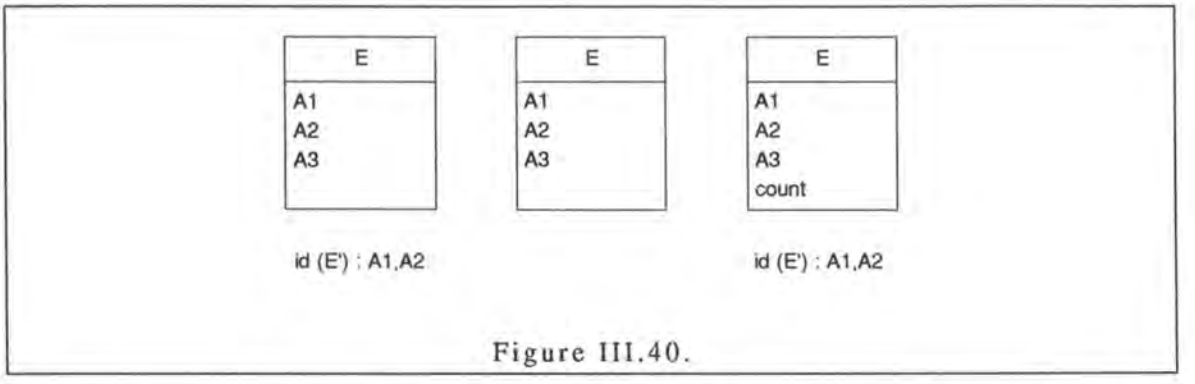


Figure III.40.

Dans la situation où EA2 n'a pas d'identifiant, un index sur A2 nous indiquera avec un bon niveau de certitude que l'ordre a de l'importance.

Mais, dans les deux autres cas où A2 participe à l'identifiant, seul le fait que l'index soit déclaré sur le couple (A2,A1) et non pas sur le couple (A1,A2) pourait eveiller notre attention. Nous admettons que cet indice n'est pas des plus fin et que son niveau de certitude est faible.

Enfin, une vue basée sur EA2 où la commande de tri est présente et porte sur A1 et A2 peut être considérée comme étant une preuve que la notion d'ordre doit être prise en compte.

3.4.3.4.3. LES TECHNIQUES DE L'INSTANCIATION ET DE LA CONCATÉNATION

En ce qui concerne ces deux technique, les traces d'une notion d'ordre ne sauraient être trouvées que par l'analyse des données.

Pour la technique d'instanciation, on remarquera un ordonnancement des valeurs dans la suite des attributs.

Quant à la technique de concaténation, l'ordonnacement se marquera dans la suite des sous-ensembles de l'attribut.

3.4.4. LA RECHERCHE DES TYPES D'ASSOCIATION

Dans le modèle relationnel, pour mettre en association deux relations, on définit dans l'une une clé étrangère (foreign key). La valeur de cette clé étrangère pour un tuple donné doit être identique à une valeur de clé primaire pour un tuple de l'autre relation. De cette manière, les deux tuples sont en association l'un avec l'autre tout comme deux entités peuvent être liées par une association.

Notre recherche des types d'association pourait donc porter sur l'étude des clés étrangères et des clés primaires. Mais noublions pas que ces dernières ont pu être choisies parmis plusieurs clés candidates donc notre étude s'élargira à toutes les clés candidates et toutes les clés étrangères qui les référencient. Cette option avait déjà été prise au niveau de la phase d'extraction des structures de données en élargisant les concepts de clé primaire et étrangère.

Après cette mise au point, nous pouvons décrire les renseignements sur lesquels se base notre recherche de type d'association. Cette description est reprise à la figure III.41.

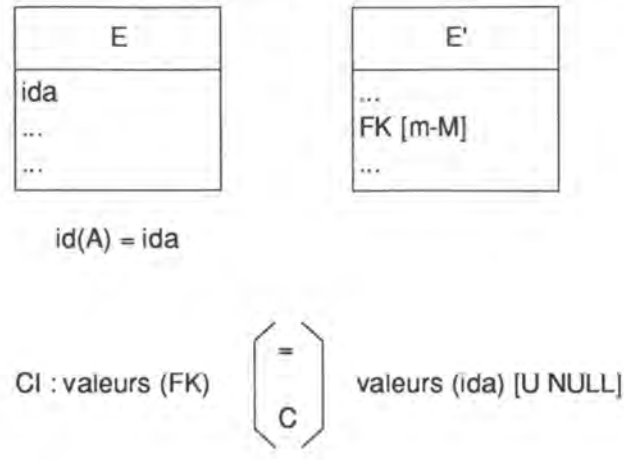


Figure III.41.

Les renseignements que nous fournit la phase d'extraction des structures de données nous permettent de savoir qu'une relation A et une relation B sont associées via la clé candidate "ida" dans la relation A et la clé étrangère "FK" dans la relation B. De plus, nous serons au courant de la contrainte d'intégrité qui accompagne cette mise en association. Cette contrainte aura été validée et nous apprendra le rapport qui existe entre les valeurs prises par la clé candidate et celles prises par la clé étrangère.

Avant d'expliquer la manière de rechercher le type d'association, décrivons exactement le but assigné a cette recherche.

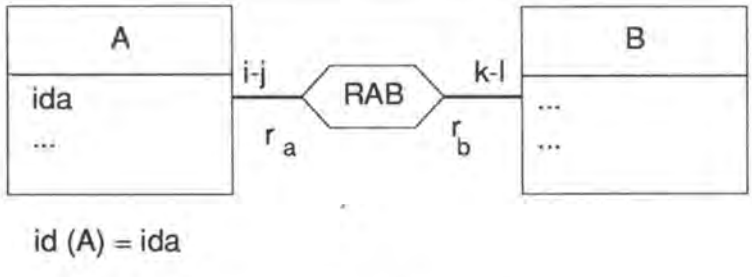


Figure III.42.

Le but de la recherche est d'exprimer un type d'association RAB liant les types d'entité A et B respectivement représentés par la relation A et la relation B de la figure III.41. De plus, il nous faudra connaître les connectivités minimum et maximum des rôles r_a et r_b que jouent les types d'entité A et B dans le type d'association RAB.

Maintenant que les tenants et les aboutissants de la recherche sont clairement décrits, voyons comment celle-ci se déroule.

Dans un premier temps, la relation RAB sera exprimée ainsi que les rôles joués par les deux types d'entité. Un nom sera donné à ces trois objets.

Ensuite nous procéderons à la recherche des différentes connectivités représentées à la figure III.42. par les lettres "i", "j", "k" et "l".

La valeur de "i" dépendra du rapport qui existe entre les valeurs de FK et les valeurs de "ida". Si l'ensemble des premières est inclu dans l'ensemble des secondes alors "i" vaudra 0. Par contre si ces deux ensembles sont égaux, alors "i" vaudra 1. De fait, si une valeur de l'identifiant ida peut ne pas être reprise dans l'ensemble des valeurs de FK, cela signifie qu'un type d'entité A peut ne pas jouer un rôle dans RAB.

La valeur de "j", quant à elle, vaudra 1 si FK est un identifiant potentiel du type d'entité B et n dans le cas contraire. Nous expliquons cette règle par le raisonnement suivant : Si FK est un identifiant du type d'entité B, alors cela veut dire qu'à une valeur de FK correspond au plus une entité de type B. Comme nous savons qu'une valeur de FK est aussi une valeur de "ida", nous pouvons également dire que si FK identifie le type d'entité B alors à une valeur de "ida" correspond au plus une entité de type B. D'où la connectivité "j" vaut 1.

Enfin les valeur des connectivités "k" et "l" sont respectivement égales aux valeurs de "m" et "M" issues de la figure III.42. qui expriment le caractère obligatoire et multivalué de la clé étrangère dans le type d'entité B.

Notons également que si "j" vaut 1 le type d'entité A est identifié par le rôle qu'il joue dans le type d'association RAB. Et symétriquement, si "l" vaut 1 alors le type d'entité B est également identifié par son rôle dans RAB.

3.4.5. CONCLUSION

Tout au long de ce chapitre, nous avons découvert des méthodes pour exprimer le schéma de la base des données avec des concepts d'un plus haut niveau d'abstarction que les simples concepts de bases du modèle relationnel. Le résultat obtenu après application de ces méthodes sera un schéma conceptuel simplifié exprimé dans le modèle Entité-Association.

A ce niveau-ci, on peut dire que nous sommes arrivé au terme du processus de rétro-ingénierie. Toutefois, certains utilisateurs ne se contenteront pas de ce schéma conceptuel simplifié. De ce fait nous vous proposons la lecture du chapitre 3.2., qui concerne la recherche de concepts de niveau d'abstarction encore supérieur.

4. RÉTRO-INGÉNIERIE SUR BASES DE DONNÉES CODASYL

4.1. QU'EST QU'UN SGBD CODASYL ?

4.1.1. GÉNÉRALITÉS SUR CODASYL

Le modèle CODASYL est le modèle le plus proche du modèle Entité-Association.

Ses principales caractéristiques sont les relations 1-N, non récursives, un seul identifiant, une seule clé secondaire par entité, par de clé d'accès répétitive et la possibilité d'avoir des attributs multivalués.

Les concepts CODASYL sur lesquels nous basons la première partie de notre réflexion sont au nombre de 8:

- les *areas* (collections d'entités).
- les *records* (types d'entité),
- les *data* (attributs), décomposables et/ou répétitifs, mais jamais facultatifs.
- les *sets* (types d'associations), contenants un *owner* et plusieurs *members*,
- les *owners* (rôles 0-1, 1-1, 0-N ou 1-N),
- les *members* (rôles 0-1 ou 1-1),
- les *absolute identifiers*, déclarés dans la clause *location mode* de la description du *record*,
- les *relative identifiers*, déclarés dans la clause *member* de la description du *set*.

Les types d'accès aux *records* au nombre de trois, sont définis dans la description des *records*, et plus précisément dans la clause ***LOCATION MODE*** :

- Accès calculé par une clé.

Cette clé est constituée d'un ensemble de *data* contenus dans ce *record*, dont la duplicité est admise ou non par la phrase **duplicate** allowed ou **duplicate not** allowed.

Exemple : ***location*** mode is **calc using** ***data***_name[, *data*_name2[...]] **duplicates** are [**not**] allowed.

- Accès via un *set* (accès à un *member* à partir d'un *owner*).

Exemple : ***location*** mode is **via** ***SET***_name ***set***.

- Accès direct , dont la clé doit être déclarée comme une *Data Base Key*.

Exemple : ***location*** mode is **direct** ***DATA*_BASE_KEY**_name.

4.2. OÙ TROUVER L'INFORMATION ?

Le processus, de rétro-ingénierie, visant à reconstruire le schéma conceptuel, s'appuie sur un certain nombre de sources d'informations, dont la consultation augmentera le degré de certitude des hypothèses émises tout au long du raisonnement.

4.2.1. LE SCHÉMA GLOBAL

Une simple lecture de la description d'un schéma CODASYL peut fournir les informations suivantes :

- Entités (nom, clé primaire et/ou secondaire)
- Attributs(nom, décomposable, répétitif, type)
- Associations(nom, rôles(nom, cardinalités, critère de tri))

==> Première version du schéma conceptuel (schéma conceptuel brut).

D'après nous, seul le DDL CODASYL est utile au RI, du moins dans la partie visant à reconstruire le schéma conceptuel. Les autres informations décrivent physiquement l'organisation des données (schéma DMCL et SSL CODASYL par exemple). Le plus souvent, ces descriptions sont dépendantes d'une configuration informatique bien précise. C'est pour ces raisons que ces spécifications ne nous semblent pas intéressantes.

4.2.2. LES SOUS-SCHÉMAS (EN-TÊTES COBOL)

- Affinement du schéma global, par une redéfinition plus détaillée , ou changement de nom des *data* des niveaux supérieurs à 2.

==> Schéma conceptuel brut affiné.

4.2.3. LE CODE CONTRÔLANT OU EXPLOITANT LES SOUS-SCHÉMAS

Nous entendons par code de contrôle, toutes les procédures attachées au schéma, et chargées de conserver l'intégrité de celui-ci, par l'entremise du SGBD. Le code d'exploitation, est l'ensemble des listings des applications utilisants le ou les sous-schéma.

- Contraintes d'intégrité statiques, dynamiques, d'inclusion ou référentielles, structurelles non exprimées dans le schéma CODASYL global.
- Affinement du schéma global par mise en évidence de mouvement de champs vers des zones mémoires plus détaillées, et inversement.

==> schéma conceptuel brut affiné et complété.

4.2.4. L'ANALYSE DES DONNÉES ELLES-MÊMES

Cette analyse permet de mettre en évidence :

- Le sous-typage (sous forme agrégée d'entités) par l'exclusion de certaines valeurs d'attributs entre elles, au sein d'une même entité.
- Le domaine de valeurs des *data*, leur type et leur structure réelle.
- Les dépendances fonctionnelles, les identifiants.

4.3. EXTRACTION DES STRUCTURES DE DONNÉES

4.3.1. EXTRACTION DES INFORMATIONS DU SCHÉMA GLOBAL

La première étape du processus d'extraction des données, consiste à traduire le schéma CODASYL en un schéma Entité-Association équivalent, de niveau logique. Ce schéma EA sera par la suite enrichi notamment par la consultation de ses sous-schémas COBOL et des listings d'applications exploitants ces sous-schémas.

Afin de passer d'un mode de représentation à un autre (CODASYL => Entité-Association), il est nécessaire d'utiliser un importateur de schémas, qui va, en quelque sorte, traduire la description de la base de données CODASYL, en concepts propres au modèle EA. Notons au passage que l'information qui ne peut être directement exploitée par l'importateur est mise de côté afin de pouvoir être utilisée ultérieurement. De plus, l'entièreté des informations non utilisées peut être rappelée à n'importe quel moment du processus de rétro-ingénierie.

Ces clauses, tirées du manuel CODASYL IDS-II **[Bul83]**, figurent ici pour illuster les possibilités du modèle.

Les traductions effectuées sont les suivantes :

4.3.1.1. LES COMMENTAIRES

Dans la description d'un schéma CODASYL, les commentaires peuvent se retrouver quasiment à n'importe quelle place. L'idéal, lors de la rencontre de l'un de ceux-ci serait de l'attacher directement au concept EA concerné. Ces commentaires, contenants la plupart du temps de l'information informelle, donc non-exploitable automatiquement, pourraient fournir ultérieurement des renseignements à l'utilisateur du CARE

4.3.1.2. LE NOM DU SCHÉMA

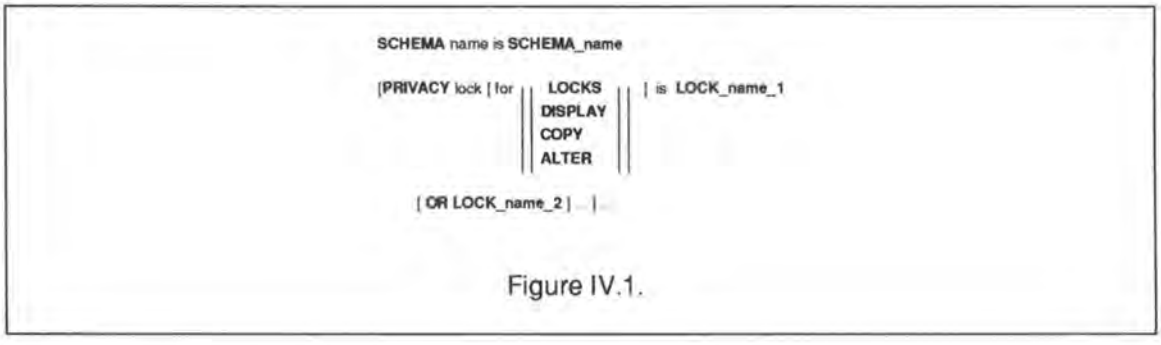


Figure IV.1.

- Le nom du schéma CODASYL figure dans la première ligne de description de la base de données CODASYL. C'est ce nom que l'on donnera au nom du schéma EA.
- La clause *Privacy* (facultative) suivant la clause schema-name, renseigne sur les locks établis sur le schéma lui-même.

4.3.1.3. LES *AREAS*

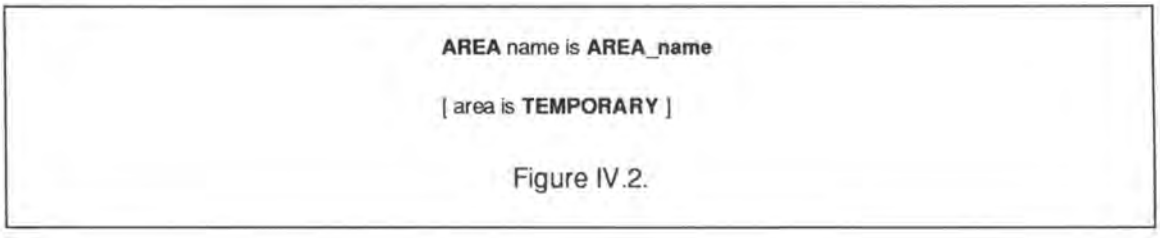


Figure IV.2.

CODASYL offre la possibilité de définir nombre *d'areas*, qui sont des fichiers pouvant regrouper plusieurs *records*. Au fur et à mesure de la traduction des *records*, on rattachera ceux-ci à la (aux) collection(s) d'entités correspondante(s).

4.3.1.4. LES *RECORDS*

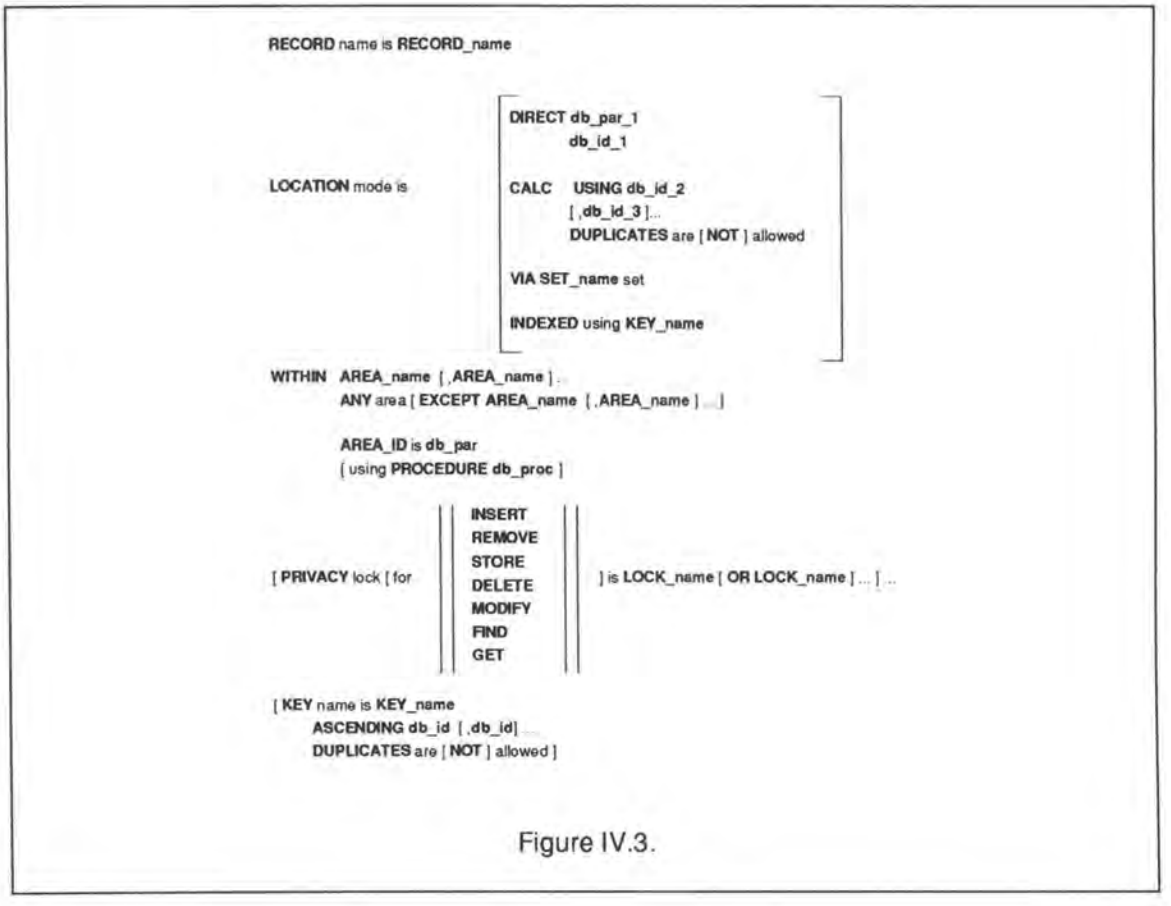


Figure IV.3.

- La clause ***RECORD*** fournit le nom de l'entité.

- La clause ***LOCATION*** donne le moyen d'accès au *record* les cas suivants sont possibles :

*Accès direct ou via le *set*, cette clause est laissée de côté.

*Accès calculé, l'ensemble des champs énumérés fournit les attributs de l'entité formant la clé d'accès. Si les doublons ne sont pas admis (duplicates not allowed), cette clé est un identifiant. Cette version de la clause sera traduite par la création d'une clé identifiante ou non, selon le cas.

*Accès indexé, la description de la clé d'index se trouve alors dans la clause Key.

- La clause **WITHIN** énumère les *areas*. Pour chaque *area* rencontrée, on construit la liste des *records* qui la concerne. Traduites, elles deviennent des collections d'entités.

- La clause ***PRIVACY*** renseigne les actions sur le *record* pour lesquelles un lock est effectué. Cette clause est entièrement laissée de côté.

- La clause **KEY** (facultative) n'est présente que si le *location* mode est indexé. L'ensemble des champs énumérés est traité comme pour l'accès calculé.

4.3.1.5. LES *DATA*

En plus des renseignements sur le *record*, la clause *record* est constituée d'une liste de *data*. Pour chacun de ces *data*, on trouve :

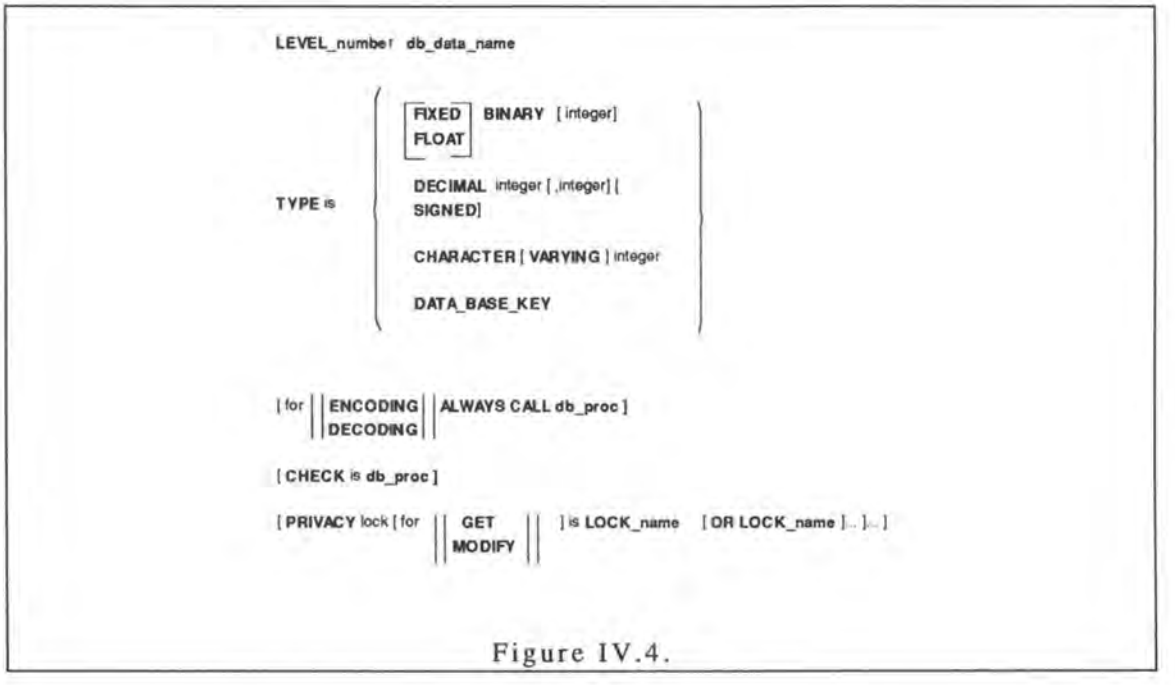


Figure IV.4.

- Une clause **TYPE**, qui fournit une description physique du *data*. Chaque type sera repris tel quel. Certains SGBD CODASYL permettent de définir des *data* répétitifs au sein d'un *record*. Si par exemple le *data* est défini *OCCURS* 4, par défaut, nous attribuerons les cardinalités minimum et maximum égales à 4 pour l'attribut du type d'entité.

- Une clause ***ENCODING-DECODING*** (facultative), qui assigne une procédure pour l'encodage, la modification ou la lecture d'un *data* (actions get et store). Cette information n'est pas directement exploitable via l'importateur, mais elle sera consultée lors de l'analyse des listings sources, afin de dépister d'éventuelles contraintes d'intégrités.

- Une clause ***CHECK*** (facultative), ayant le même rôle que la clause *encoding-decoding*, et qui est exécutée (si les deux sont déclarées) après. Cette clause n'entre en action que pour l'écriture et la modification de *data*.

- Une clause ***PRIVACY*** (facultative), qui établit un lock pour la lecture et/ou la modification.

Exemple:

```
SCHEMA NAME IS BUCLIDS.

    RECORD NAME IS CLIENT
      LOCATION CALC USING NUM-CLI DUPLICATES NOT
      WITHIN AREA1.
    02  CLI-NUM        TYPE CHARACTER 12
      FOR ENCODING DECODING ALWAYS CALL GET-CLI-NUM.
    02  CLI-ADRESSE.
     03 RUE            TYPE CHARACTER 20.
     03 VILLE          TYPE CHARACTER 25.
    02 CLI-TEL         OCCURS 4.
     03 NUMERO           TYPE CHARACTER 15.

  END-SCHEMA.
```

FIGURE IV.5.

- Cette description CODASYL sera traduite en EA de manière suivante :

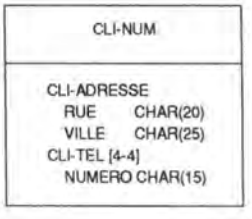


Figure IV.6.

4.3.1.6. LES *SETS*

La description d'un *set* comporte deux parties principales, la définition de l'***OWNER***, et la définition du ou des ***MEMBERS***.

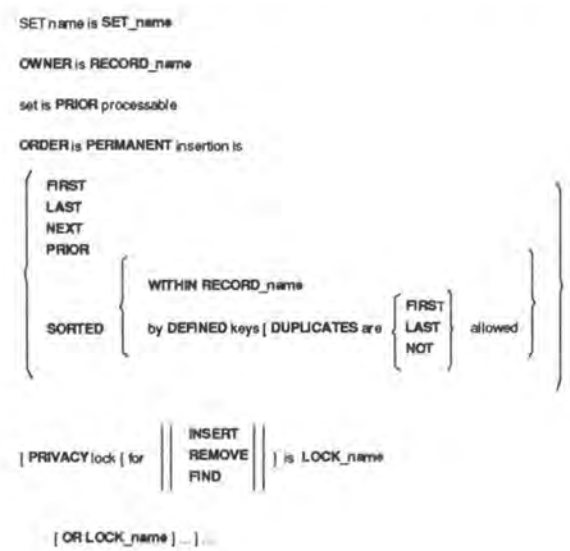


Figure IV.7.

La première partie se compose de 5 clauses.

- La clause ***SET***, qui lui attribue un nom. La rencontre de cette clause donne lieu à la création d'un type d'association du même nom.

- La clause ***OWNER***, qui définit le *record* "propriétaire". Cette clause donne lieu à la création d'un rôle reliant l'entité

concernée et l'association courante. Dans le modèle EA, les rôles ont un nom. On pourrait, entre autres solutions, construire un nom composé à partir du nom de l'entité et de l'association, pour notre part, nous avons choisi de donner le seul nom de l'entité au rôle.

- La clause ***PRIOR*** qui est obligatoire et non paramétrable, donc inintéressante. Cette clause ajoute un pointeur aux *records*, rendant le traitement des *sets* plus performants dans les deux sens de l'accès (*owner* ⇔ *member*) .

- La clause **ORDER**, définit le point d'insertion d'une nouvelle occurrence de l'*owner*. La cardinalité minimale est 0, toutes les occurrences ne devant pas forcément participer au *set*. L'instruction duplicates, indiquera la cardinalité maximale. Si cette instruction est suivie de NOT, la cardinalité maximum sera 1. dans les autres cas, elle sera N.
  La clause indique aussi, l'ordre d'accès des *records* par le *set*. Cette information sera conservée pour éventuellement en retenir plus tard des indices d'index ou de critères de listes.

- La clause ***PRIVACY*** (facultative), définit des lock sur le *set* pour les opérations d'insertion, d'effacement et d'accès (*find*).

La seconde partie, la définition du ou des *members* se compose de 2 clauses.

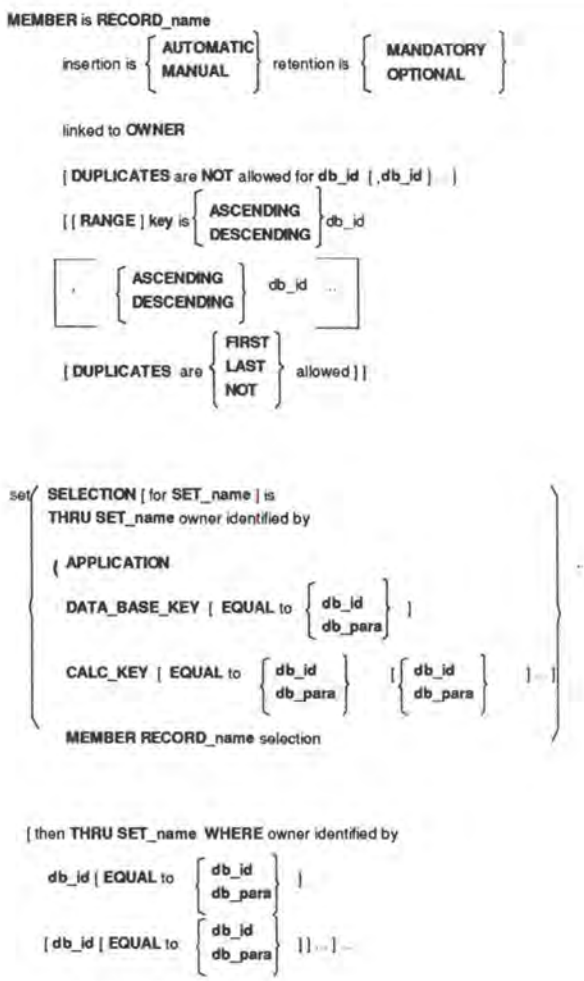


Figure IV.8.

- La clause ***MEMBER*** elle-même, qui fournit le nom du *record* concerné. Comme pour l'*owner*, elle donne lieu à la création d'un rôle, entre l'association et l'entité, qui aura pour nom le nom du *record*. Mandatory, par opposition à optional, signifie que occurrence d'un *record* ne peut être retirée, mais seulement ré affectée.

Automatic, par opposition à manual, indique que l'insertion s'effectue toujours par une procédure attachée au schéma. Si retention est mandatory, et insertion automatic, alors la cardinalité minimale sera 1.

Si retention est optionnal et insertion manual, la cardinalité minimale sera 0.

Toute autre configuration, aura par défaut la cardinalité minimale égale à 0, la décision finale appartenant à l'opérateur. La phrase duplicates, énumère un sous-ensemble de *data* du *record*,ne pouvant prendre deux fois la même valeur pour une valeur donnée de l'*owner*. La phrase duplicates, si elle est mentionnée, fournira ultérieurement des indices sur un identifiant secondaire.

- La clause **SELECTION**, définit les règles qui gouvernent la selection de l'occurrence appropriée d'un *set*, dans le but d'insérer ou d'accéder à un *member*. Cette clause n'est pas traduisible en concepts EA.

Exemple : Soit la description du *set* SMZKETTE suivante,

```
SCHEMA NAME IS BUCLIDS.
(...)
   SET NAME IS SMZKETTE
      OWNER IS SPARKONTO
      ORDER PERMANENT
         SORTED BY DEFINED KEYS
       DUPLICATES FIRST.
   MEMBER IS SSPERRE
      INSERTION AUTOMATIC RETENTION MANDATORY
      DUPLICATES NOT FOR SSSPS
      KEY IS ASCENDING RECORD-TYPE DESCENDING SSSPS
      SET SELECTION IS THRU SMZKETTE BY APPLICATION.
   MEMBER IS SHINWEIS
      INSERTION AUTOMATIC RETENTION MANDATORY
       DUPLICATES NOT FOR SHKONTO
      KEY IS ASCENDING RECORD-TYPE SHKONTO
      SET SELECTION IS THRU SMZKETTE
        OWNER IDENTIFIED BY CALC-KEY.
(...)
END-SCHEMA.
```

FIGURE IV.9.

La représentation EA correspondante sera :

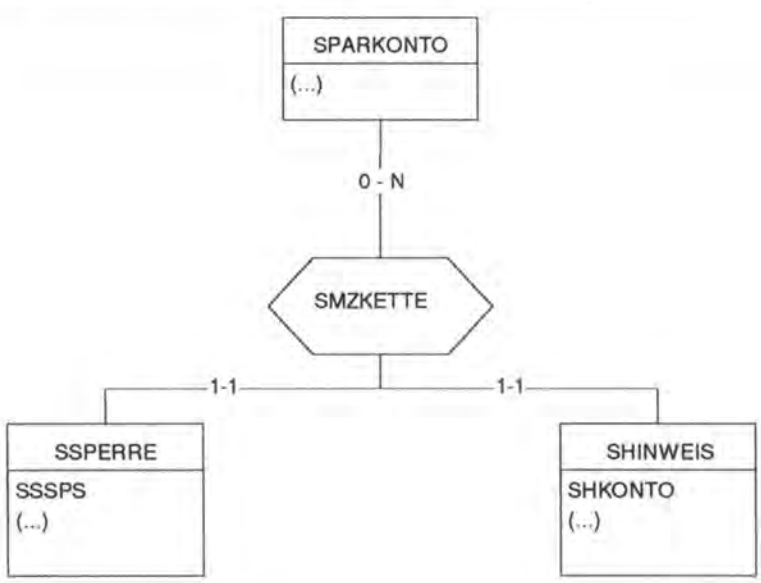


Figure IV.10.

Remarque L'information non traduite reste accessible pour les phases ultérieures de la tâche de RI.

4.3.2. EXTRACTION DES INFORMATIONS DES SOUS-SCHÉMAS CODASYL

Comme dans la plupart des SGBD's, CODASYL offre la possibilité de n'autoriser l'accès qu'à une partie seulement du schéma global. La description de cette vue, est un sous-schéma, figurant en en-tête de l'application COBOL.

Dans notre raisonnement, nous supposons que les schémas consultés sont syntaxiquement corrects, puisqu'ils sont utilisés réellement par des applications.

Chaque sous-schéma du schéma global, donnera lieu à la création d'un schéma EA, référençant le schéma EA du schéma global. Les informations modifiées d'un sous-schéma feront référence aux informations concernées du schéma EA du schéma global.

Notons que L'analyse d'un sous-schéma n'apporte pas d'information fondamentale. Elle permet tout au plus d'affiner la définition des *records* du schéma global.

L'analyseur de schéma CODASYL, effectue le lien entre un schéma global et ses sous-shémas, par comparaison des noms de *records* et des zones 02 du schéma global d'une part, et les noms de zones 01 et 02 du sous-schéma d'autre part.

L'analyse des sous-schémas peut se faire pour deux raisons distinctes. La première, consiste à affiner la définition d'un schéma global grâce aux différentes vues qu'offrent les sous-schémas. La seconde, tente, lorsque la définition du schéma global n'est pas disponible, de reconstruire une vue d'ensemble à partir des sous-schémas. Notons que cette dernière éventualité est très rare en CODASYL.

Remarque: La définition d'un sous-schéma CODASYL offre la possibilité de renommer des objets du schéma global (*records*, *sets*, *areas*, *data*, keys). Le mécanisme de cette redéfinition peut s'apparenter aux *# define* du langage C. Dans l'*ALIAS SECTION*, on donne une série de noms d'objets du schéma global, et pour chacun de ceux-ci, le nom qui lui correspond dans le sous-schéma. Cette *section* du code, est le lien entre les noms donnés dans le schéma global, et ceux utilisés par le sous-schéma et donc par l'application. L'ensemble des informations de cette *section* sera conservé sous forme de synonyme pour chaque objet concerné.

4.3.2.1. AFFINEMENT DU SCHÉMA GLOBAL

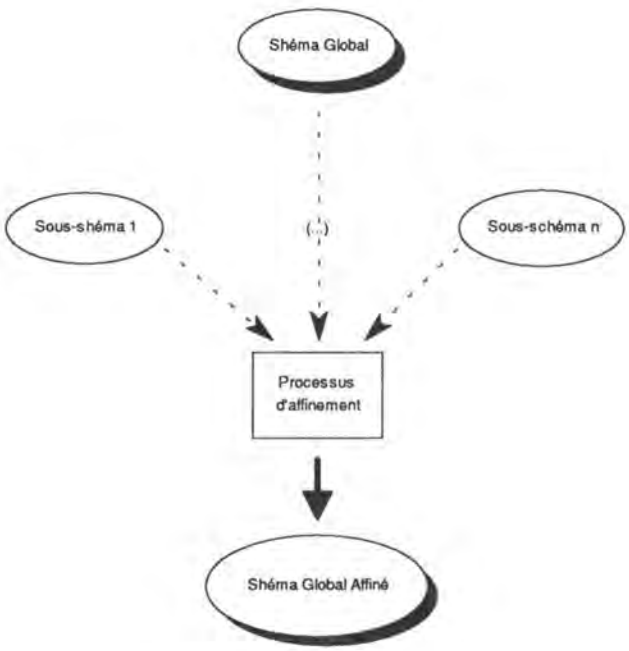


Figure IV.11.

Préconditions :

- Le schéma global existe,
- le schéma global et les sous-schémas sont syntaxiquement corrects.

La première étape consiste à traduire les sous-schémas en schémas EA équivalents. Ici, le niveau 01 donne les types d'entité, les niveaux directement inférieurs les attributs, et les autres niveaux, la composition des attributs.

Pour chaque sous-schéma EA (SSEA), on établit un lien vers le schéma EA global (SEAG). En comparant chaque entité du SSEA avec l'entité correspondante du SEAG, on peut détecter les redéfinitions de zones. Deux situations peuvent alors se présenter.

4.3.2.1.1. DESCRIPTIONS COMPLÉMENTAIRES

C'est le cas le plus simple, lorsque la définition du sous-schéma affine la définition du schéma global. Dans ce cas, il y a concordance des noms de *data*.

Il suffit alors de mettre à jour le schéma global EA en liant les deux définitions d'attribut entre-elles.

Exemple :

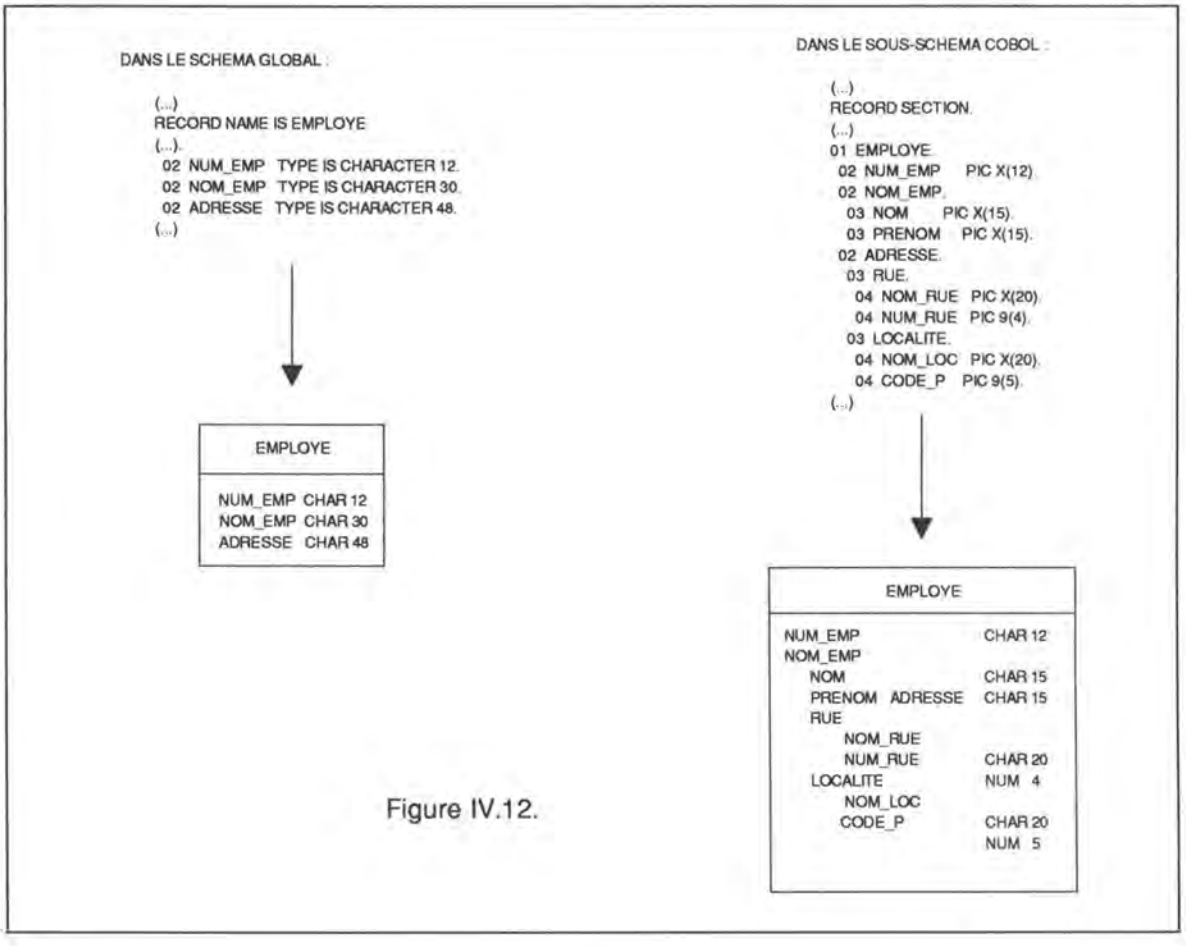


Figure IV.12.

4.3.2.1.2. CONFLITS DE STRUCTURES ENTRE SOUS-SCHÉMAS

Ce type de problème se rencontre lorsque plusieurs sous-schémas présentent un découpage différent d'un attribut du SEAG.

Le cas typique est de trouver deux vues dont les définitions se chevauchent.

La question qui se pose est de savoir quelle vue adopter. Sachant que les vues sont d'importance équivalentes, il nous semble juste soit de définir l'attribut du SEAG en question, de manière plus générale, et de fournir les vues sous forme de commentaires (synonymes par exemple), soit d'en privilégier une d'entre elles et de placer les autres en commentaire. Le critère de sélection de la vue à retenir est bien sûr, en dernier ressort, la tâche de l'opérateur, mais les solutions suivantes sont possibles.

- Choisir la structure qui se retrouve le plus fréquemment parmi les sous-schémas.

- Choisir la structure dont la définition est la plus évidente (intuitive).

- Choisir la structure qui offre le découpage le plus clair.

- Choisir la structure du schéma global quand aucune des trois autres solutions n'est préférée.

Notons toutefois que ce cas est relativement rare, car peu de données peuvent offrir de multiples interprétations.

Exemple :

Dans le schéma global :	Dans un premier sous-schéma :	Dans un second sous-schéma :
02 LISTE type is CHARACTER 16.	02 LISTE pic X(8) occurs 2.	02 LISTE pic X occurs 16.

Figure IV.13.

4.3.2.2. RECONSTRUCTION DU SCHÉMA GLOBAL

Jusqu'alors, le processus de RE, s'est intéressé à la reconstitution du schéma conceptuel du système d'information. Ce processus s'est effectué sur base d'un schéma global, affiné par ses différentes vues que sont les sous-schémas, quand il y en a.

Une autre démarche est cependant possible, qui ne requiert pas l'existence d'un schéma global. Pour obtenir un nouveau schéma global aussi complet que l'original, nous devons admettre que l'ensemble des sous-schémas disponibles recouvrent l'entièreté du schéma global.

A défaut, nous obtiendrons un nouveau schéma global reflétant la structure de l'ensemble des sous-schémas.

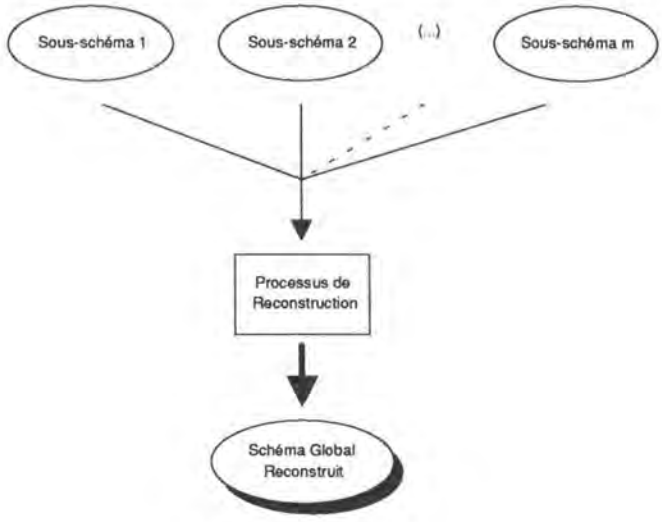


Figure IV.14.

les conflits de structures sont les mêmes que ceux décrits lors de l'affinement du schéma global.

4.3.3. EXTRACTION DES INFORMATIONS DU CODE CONTRÔLANT OU EXPLOITANT LE SYSTÈME D'INFORMATION

4.3.3.1. LES PROCÉDURES DE CONTRÔLE

CODASYL, comme bon nombre de SGBD, offre la possibilité de lier certaines procédures à la gestion du schéma, ce sont ce que nous appellerons, les procédures de contrôle. Ces procédures sont généralement l'expression d'une contrainte destinée à conserver l'intégrité des informations en autorisant ou non l'ajout ou la modification de certaines valeurs de données du schéma. Dans le cas CODASYL IDS-II dont la syntaxe à été donnée ci-dessus, il s'agit de la clause *check* des *data*.

La consultation et l'interprétation de ce code, peut fournir des indices sur le domaine des *data* du schéma.

Les contraintes d'intégrité peuvent être de deux types; statiques ou dynamiques.

Les premières définissent un ensemble de valeurs possibles, en extension (énumération des valeurs) ou en compréhension (définition d'une propriété que la valeur doit respecter). Dans le cas d'une contrainte d'intégrité dynamique définie en extension, il se pourrait que cette dernière fasse référence à un ensemble de valeurs prises par un autre attribut, il s'agit alors d'une contrainte d'inclusion, ou d'une contrainte référentielle.

Dans la classe des contraintes d'intégrité statiques, on peut ajouter les contraintes 'structurelles', qui ne s'intéressent pas à la valeur proprement dite du *data*, mais plutôt à sa structure.

Les secondes, dynamiques, définissent de quelle manière les valeurs d'un attribut doivent évoluer. Citons par exemple le cas d'un attribut prix, dont la valeur modifiée ne pourrait être que croissante.

Comme nous le voyons, il y a une gradation dans la complexité des contraintes, dans la manière de les exprimer et bien sûr, dans la façon de les mettre en évidence.

Il va sans dire, que l'analyse du code est une chose très complexe, et nous n'aborderons, dans cette partie, que des aspects simplifiés du problème.

Pour étendre le raisonnement, nous ne nous attacherons pas à une syntaxe particulière d'un SGBD CODASYL. Admettons qu'une procédure *check* se présente sous la forme d'une fonction ayant comme paramètre l'ancienne et la nouvelle valeur d'attribut et qui renvoie ou non l'autorisation de l'opération d'ajout ou de mise à jour.

4.3.3.1.1. CONTRAINTES D'INTÉGRITÉ STATIQUES

Ce sont les contraintes les plus faciles à mettre en évidence. Elles sont représentées par une série de tests, dont un des membres est la variable contenant la nouvelle valeur du *data*. Notons que la contrainte statique ne se préoccupe pas de l'ancienne valeur du *data*. Donc, si on ne trouve pas trace de manipulation de la variable contenant l'ancienne valeur du *data*, il y de fortes chances qu'il s'agisse d'une contrainte statique.

Exemple de contrainte d'intégrité statique :

```
CHECK for data AGE of record-type PERSONNE
   old value is OLD_val.
   new value is NEW_val.

      IF NEW_VAL > 0 and NEW_VAL<120
         return(OK)
      END-IF.
END-CHECK.
```

Figure IV.15

Les contraintes structurelles, sont un peu plus compliquées à repérer, car leur contrôle peut s'effectuer de plusieurs manières différentes. Cette distinction peut se faire sur base du type de *data*.

Par exemple :

- pour un *data* de type nombre dont on veut qu'il commence toujours par 999.

- pour un *data* de type CHARACTER dont la 3ième et 7ième position doit-être un '\'.

La recherche de structure au sein de *data*, apporte de l'information au niveau sémantique, et permet par exemple de reconnaître qu'un *data* contient en réalité un numéro de compte ou un numéro de téléphone.

Pour détecter ces contraintes structurelles, il serait utile d'analyser les procédures indiquées par la clause *ENCODING-DECODING* de la description d'un *data*. Cette clause effectue un traitement sur la présentation des données introduites dans la base par une application, et inversement, lorsque ces données vont de la base vers l'application. Leur implémentation provient sans doute d'un soucis d'économie de place (certains caractères constamment au même endroit dans tous les *data* représentent une perte de place) et de temps nécessaires à ce traitement de présentation des *data*.

Figure IV.16.

L'analyse la plus efficace de la structure d'un *data* se fera dans la partie X.4., lors de l'analyse des données elles-mêmes.

4.3.3.1.2. CONTRAINTES D'INTÉGRITÉ DYNAMIQUES

Comme nous l'avons dit plus haut, ces contraintes définissent plus ou moins étroitement l'évolution des valeurs du *data*. Pour effectuer ce contrôle, il est donc nécessaire de manipuler la variable contenant l'ancienne valeur. Si on recherche particulièrement les contraintes dynamiques, c'est la présence de cette variable dans les tests qui en sera l'indice.

Exemple de contrainte d'intégrité dynamique :

```
CHECK for data PRICE of record-type ARTICLE
   old value is OLD_P.
   new value is NEW_PI.

      IF NEW_P >= OLD_P * 1.25
        return(OK)
      END-IF.
END-CHECK.
```

Figure IV.17.

4.3.3.1.3. CONTRAINTES D'INCLUSION ET CONTRAINTES RÉFÉRENTIELLES

Le troisième type de contrainte décelable dans les clauses *CHECK*, sont celles qui définissent les valeurs en termes de référence aux valeurs d'un autre *data*.

Si l'on s'interroge sur la raison d'implémentation de ces contraintes, on peut remarquer deux choses. La première, c'est l'intention délibérée du concepteur, de limiter l'ensemble des valeurs prise par ces *data* par une inclusion dans le domaine des valeurs d'un autre *data*. La seconde, révèle l'implémentation de ce que l'on pourrait appeler une foreign key, ou en d'autre terme une référence vers un autre *record*. Ce *data* de référence pourrat-être ultérieurement transformé en relation entre les deux *records*.

Il existe une manière de déceler si un tel *data*, est l'expression d'une contrainte référentielle ou non. Si le *data* en question référence l'identifiant d'un autre *record*, il y a présomption d'implémentation d'une contrainte référentielle.

4.3.3.2. LES LISTINGS D'APPLICATION

La seconde source d'information procédurale, est l'ensemble du code exploitant le schéma global, ou le code spécifique exploitant les sous-schémas.

Les types d'informations trouvés ici sont multiples et similaires aux types d'informations déjà extraites dans le schéma global et/ou les sous-schémas.

L'analyse du code permet d'observer les manipulations qui sont effectuées directement ou indirectement sur les *records* et les *data* du schéma.

Les manipulations touchent directement les *records* et les *data* lorsqu'elles sont effectuées sur ces derniers eux-mêmes. A l'inverse, les manipulations sont indirectes, lorsqu'elles sont effectuées sur des variables contenant la valeur d'un *data*. Cette distinction nécessite une pré - analyse, qui déterminera si telle ou telle variable contient les valeurs d'un *data* lorsqu'elle est manipulée.

Les informations pouvant être retirées du code COBOL sont les suivantes :

4.3.3.2.1. AFFINEMENT DU SCHÉMA

Par observation des transferts de zones cobol vers les *data* et inversement, on peut mettre en évidence des découpages différents de la définition construite jusqu'alors.

4.3.3.2.2. CONTRAINTES D'INTÉGRITÉ STATIQUES

Par le même procédé utilisé lors de la recherche de structure dans le code de contrôle.

4.3.3.2.3. CONTRAINTES D'INTÉGRITÉ DYNAMIQUES

Par le même procédé utilisé lors de la recherche de structure dans le code de contrôle.

4.3.3.2.4. CONTRAINTES RÉFÉRENTIELLES

Par l'observation de transfert de valeur de *data* (ou de groupe de *data*) vers une KEY ou un *data* de type DBKEY

4.3.3.2.5. CONTRAINTES STRUCTURELLES

Par le même procédé utilisé lors de la recherche de structure dans le code de contrôle.

4.3.3.2.6. DATA CALCULÉS

Si un attribut, n'est l'objet d'aucune initialisation ou modification de valeur via une interface avec l'utilisateur, et que de plusil est la combinaison de plusieurs autres *data*, il est fort probable qu'il soit le résultat d'un calcul, on le commentera dans le schéma par sa formule d'obtention, si on peut la trouver en entier.

4.3.4. EXTRACTION DES INFORMATIONS DES DONNÉES

Les informations collectées par le biais de l'analyse des données sont les mêmes que celles décrites pour le SGBD SQL. Les moyens utilisés (dépendances fonctionnelles, types réels, structure de *data*, etc.,) sont également identiques.

Ce type d'extraction d'information présente une difficulté au niveau de l'accès aux données. De son côté, SQL offre l'avantage d'une interface de requête permettant un accès direct aux données. CODASYL, par contre nécessite la programmation et la compilation des procédures d'accès. Cette différence peut rendre plus fastidieuse l'analyse des données.

Les méthodes et algorithmes présentés dans cette partie ont été trouvés et testés durant notre stage à la firme BIM à Bruxelles. Ces tests ont été effectués en C et en Prolog via le SGBD Sybase.

4.3.4.3. LES ATTRIBUTS DE RÉFÉRENCES

L'analyse des données, peut être l'occasion de découvrir des relations entre types d'entité, si les autres moyens ont échoué.

Soient A,B deux types d'entité.

et a,b deux attributs tels que $a \in A$ et $b \in B$.

Figure IV.24.

Si l'opérateur est en possession d'indices laissant supposer que a est un attribut de référence vers b, tel qu'ils ont la même structure, ou que les noms sont semblables par exemple, il est alors possible de confirmer ou d'infirmer cette supposition. Les trois méthodes suivantes sont complémentaires et permettent d'obtenir des résultats fiables sans autant être sûrs.

4.3.4.3.1. MÉTHODE DES DOMAINES

La première méthode que nous proposons est immédiate et très peu sûre, elle consiste à comparer les domaines réels des deux attributs, obtenus par l'analyse des domaines et structures réels.

Si a est un attribut de référence vers b $\Rightarrow$ domaine(a) $\subseteq$ domaine(b).

Si cette propriété est vérifiée, c'est peut être le fruit du hasard, il est préférable, après obtention d'un résultat positif, d'utiliser la deuxième méthode.

4.3.4.3.2. MÉTHODE DE L'ÉCHANTILLON

Cette seconde méthode est plus sûre mais moins rapide que la première. Elle consiste à prendre une petite population de l'attribut a (par exemple 10%) et de vérifier que chaque instance de l'échantillon est inclue dans l'ensemble des instances de l'attribut b.

A la première instance de a qui ne vérifie pas cette propriété, on peut arrêter le test et affirmer que a n'est pas un attribut de référence vers b de B.

4.3.4.3.3. MÉTHODE DE LA COMPARAISON EXHAUSTIVE

Même principe que la méthode précédente, en considérant toutes les instances du type d'entité A. Ces résultats offrent une conclusion pour l'entièreté de la population de A, mais toujours aucune certitude.

4.4. CONCEPTUALISATION DÉPENDANTE DU MODÈLE

Le schéma EA obtenu jusqu'à présent, est la traduction de faits découverts dans la description du schéma global, des sous-schémas, des procédures de contrôle et de gestion et grâce à l'analyse des données elles-mêmes.

Ce schéma, a été traduit à partir de la description d'un système d'information utilisant des concepts qui lui sont propres. En forward engineering ces concepts propres peuvent être considérés comme des limitations à la traduction du schéma EA vers le modèle cible.

Ces limitations donnent lieu, dans le processus de forward engineering, à l'application de petites astuces de traduction de concepts du modèle EA en concepts du modèle CODASYL.

Les principales limitations du modèle CODASYL sont au nombre de trois : les relations 1-N, non récursives, et un seul identifiant par type d'entité.

Le but de la conceptualisation dépendante du modèle est de mettre en évidence des configurations laissant deviner de telles transformations du schéma conceptuel initial, et de tâcher de lui rendre son homogénéité.

4.4.1. IDENTIFIANTS SECONDAIRES

Nous l'avons évoqué plus haut, CODASYL n'offre pas la possibilité de définir un second identifiant pour un type de *record*.

La solution mise en oeuvre par les concepteurs de SI, est la création d'un *set*, qui permettra d'obtenir l'équivalent de l'identifiant secondaire.

"An entity type with K all-attributes identifiers will be inserted into (K-1) SYSTEM rel_types, i.e. rel_types whose (0-N) role is played by the SYSTEM entity type. Each of the (K-1) secondary ids is declared local within one of each such SYSTEM rel-type." **[Hai93]**

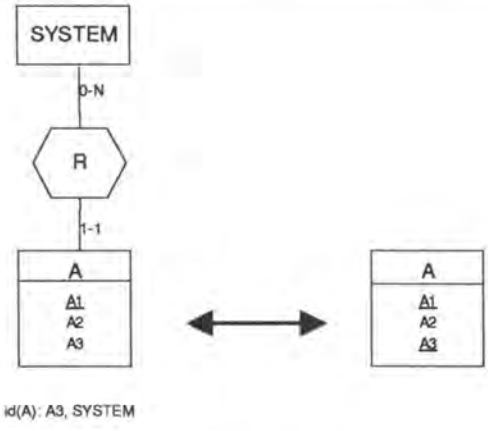


Figure IV.25.

4.4.2. ASSOCIATIONS MANY TO MANY

Lors de la transformation du shéma EA en shéma compatible CODASYL, les types de relation many-to-many sont transformées en types d'entité.

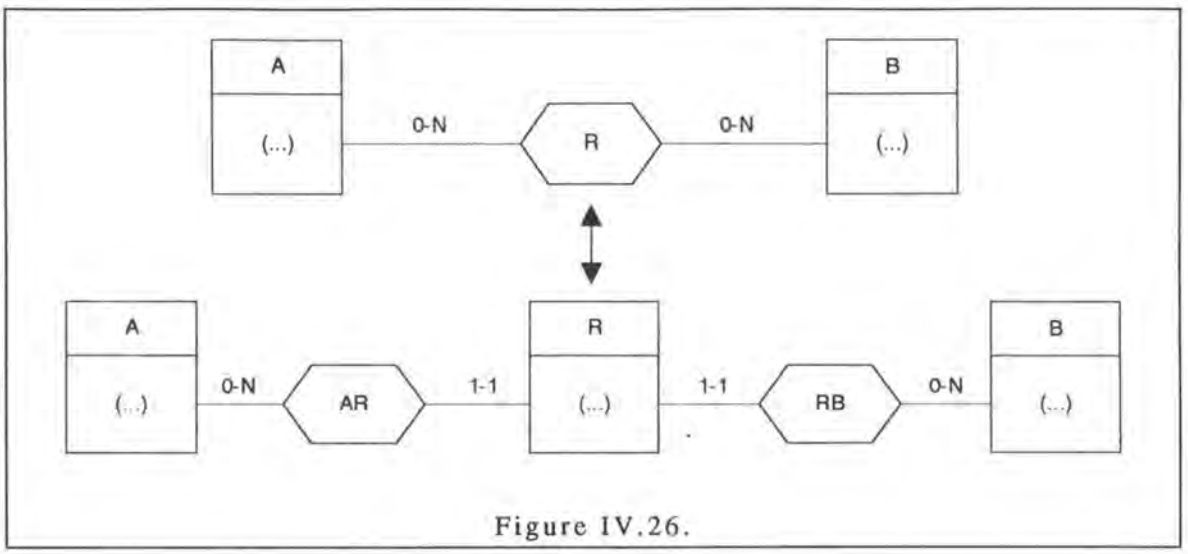


Figure IV.26.

Ces configurations peuvent soit être détectées lors de la traduction du DDL CODASYL en shéma EA, ou soit lors de l'examen du schéma EA obtenu après cette traduction.

Les particularités d'une description Codasyl représentant une many-to-many sont :

- 2 sets ayant un member commun.
- Les owners doivent accepter les 'duplicates' (sinon la cardinalité maximale serait de 1).
- Le 'record member' ne contient que deux attributs, qui sont les références (en fait les identifiants) des deux owners.

Le cas ou le type d'entité intermédiaire contiendrait plus de 2 attributs est exposé dans la partie traitant des transformations indépendantes du modèle.

4.4.3. Les préfixes

Le traitement des préfixes au sein des types d'entité, n'est pas à proprement parler une manipulation fondamentale du point de vue de la rétro-ingénierie. Il permet au plus de gagner en clarté dans la dénomination des attributs.

Lors de la conception d'une base de données, la dénomination des attributs de type d'entité, s'effectue de telle manière que l'attribut concerné puisse être facilement identifié par son nom. Pour ce faire, il n'est pas rare de lui ajouter un préfixe, rappelant le nom du type d'entité auquel il appartient, l'identifiant ainsi parmi d'autres au sein d'une application.

De tels préfixes sont inutiles dans un schéma EA, où tous les attributs d'un même type d'entité sont rassemblés. Une simple manipulation des noms d'un type d'entité au sein du schéma pourrait supprimer ces préfixes.

Exemple :

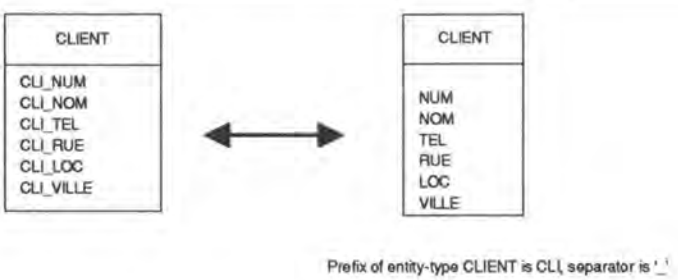


Figure IV.27.

Pré condition au traitement :

- Que tous les noms du type d'entité aient une racine commune.

Résultat de la transformation :

Cette racine est le préfixe, le reste du nom d'origine devient le nouveau nom de l'attribut

4.4.4. LES SUFFIXES

Le traitement de certains suffixes pourrait mettre en évidence l'existence d'un attribut multivalué implémenté par l'instanciation des valeurs.

Exemple. Soit le type d'entité Employé tel que :

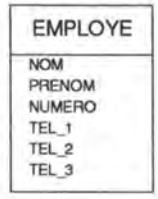


Figure IV.28.

Dans le schéma conceptuel il est probbable, que TEL soit un attribut multivalué de cardinalité 3.

La transformation que nous proposons est la suivante :

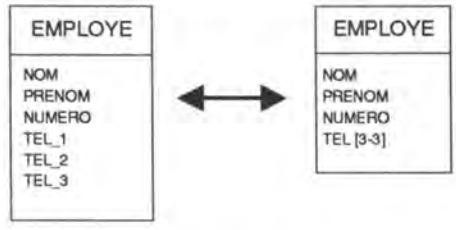


Figure IV.29.

Pré condition à la transformation :

- Plusieurs attributs ont le même préfixe.
- Ces attributs ont un suffixe composé d'un numéro.
- Ce numéro est au moins égal à i et au plus égal à j.

Résultat de la transformation :

- Les attributs en question sont supprimés.
- Ils sont remplacés par un attribut multivalué, dont le nom est leur racine commune, moins le séparateur.
- La cardinalité de cet attribut est (j-i+1).

ou

- La cardinalité est égale au nombre d'attributs remplacés.

4.4.5. RECHERCHE DES ATTRIBUTS FACULTATIFS

Dans le modèle CODASYL et contrairement au modèle Entité-Association, les attributs facultatifs ne sont pas admis. Le concepteur d'une base de données CODASYL doit donc transformer un tel type d'attribut en un concept propre au modèle CODASYL.

Deux familles de transformations sont à sa disposition. Soit il transforme l'attribut facultatif en un attribut obligatoire, soit il le transforme en une relation.

4.4.5.1. RECHERCHE D'ATTRIBUTS FACULTATIFS REPRÉSENTÉS PAR DES ATTRIBUTS OBLIGATOIRES

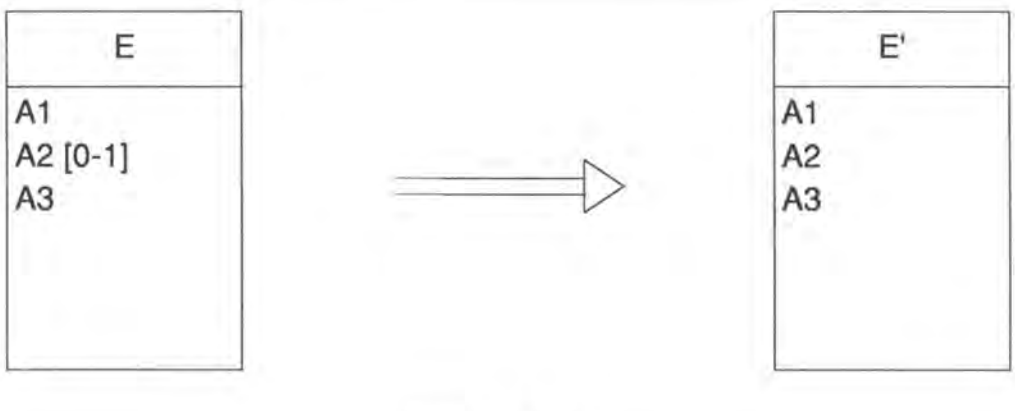


Figure IV.30.

La figure IV.30. reprend l'aspect général de la transformation qu'a subi l'attribut facultatif. Deux possibilités s'offrent alors au concepteur : Soit E'.A2 est définit sur le même domaine que E.A2, soit sur un domaine différent.

Dans le premier cas, une valeur appartenant au domaine de E.A2 (et de E'.A2) est assignée à l'attribut quand l'information n'est pas connue ou n'est pas applicable. Il va de soit que cette valeur ne pourra jamais être affectée à l'attribut comme information connue et applicable.

Dans le cadre de la rétro-ingénierie, la recherche d'un attribut ayant subit une telle transformation se base essentiellement sur la définition du domaine précis sur lequel l'attribut a été défini. Lors de l'extraction des structures de données, le domaine de chaque attribut a pu se voir affecter une valeur par défaut. Si celle-ci est comprise dans le domaine alors nous nous trouvons en face d'un attribut facultatif qui a subit la transformation décrite ici.

Dans le second cas, le plus fréquent à notre avis, il s'agit de l'ajout d'une représentation de la valeur "NULL" (étant donné qu'il n'existe pas de valeur "NULL" définie dans le modèle CODASYL) au domaine de l'attribut. La recherche de cette transformation est également basée sur le domaine.

Une seule différence intervient entre ces deux recherches : La première devra être confirmée par l'utilisateur tandis que la seconde offre un niveau de certitude tel que la confirmation peut être considérée comme superflue et ce malgré quelque cas pervers.

4.4.5.2. RECHERCHE D'ATTRIBUTS FACULTATIFS REPRÉSENTÉS PAR DES TYPES D'ENTITÉ

Comme nous le remarquerons à la vue des figures illustrant les transformations qui nous préoccupent, cette recherche ne peut s'effectuer qu'après avoir explicité les types d'association.

Deux techniques de représentation d'un attribut par un type d'entité sont envisageables. Nous allons les expliquer et étudier la manière dont on peut les retrouver.

4.4.5.2.1. LA REPRÉSENTATION PAR INSTANCE

Dans la figure IV.31. , nous pouvons observer une transformation d'un attribut en un type d'entité. Dans cette transformation, une entité représente une instance de l'attribut. C'est-à-dire, qu'il y aura autant d'entités de type EA2 que d'entités de type E qui possèdent une valeur pour l'attribut A2.

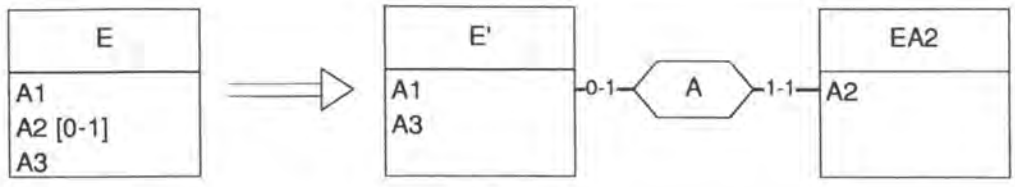


Figure IV.31.

Pour détecter le fruit d'une telle transformation, nous nous baserons sur le nombre d'attributs contenus dans les types d'entité et les connectivités posées sur les association où ces types d'entité jouent un rôle. Incontestablement un type d'entité (EA2) en correspondance avec un autre (E) via une association (A) et contenant un seul attribut (A2) peut être suspectée de représenter un attribut du type d'entité avec lequel il est associé (E). De plus, quand le rôle joué par ce dernier (E) dans l'association (A) possède une connectivité 0-1, on peut en déduire que l'attribut représenté par le type d'entité est facultatif. La connectivité 1-1 du rôle joué par le type d'entité EA2 sera la signature de la technique de représentation par instance.

Il nous suffira donc de relever dans le schéma de la base de données un couple de types d'entité associés qui vérifient ces caractéristiques.

4.4.5.2.2. La représentation par valeur

La transformation présentée dans la figure IV.32. est une variante de la précédente. En réalité, dans ce cas ci, chaque entité EA2 ne représente plus une instance de l'attribut A2 de E mais une valeur prise par cette attribut pour une ou plusieurs entités de type E. De ce fait, A2 devient identifiant de EA2 et la connectivité du rôle joué par celui-ci est 1-N.

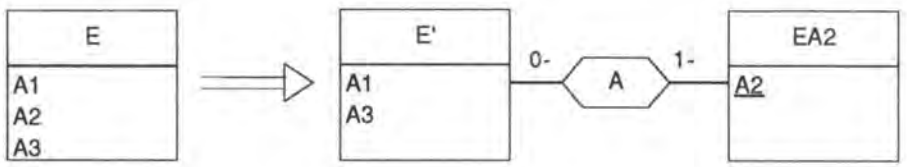


Figure IV.32.

La technique de détection de l'utilisation d'une telle transformation est identique à celle utilisée précédemment.

Notons que si E.A2 est identifiant, la connectivité du rôle joué par EA2 sera "1-1". Nous serons donc dans un cas proche de la représentation par instance, à ceci près que EA2 possède un identifiant.

5. CONCEPTUALISATION INDÉPENDANTE DU MODÈLE.

5.1. RECHERCHE DES TYPES D'ASSOCIATION REMPLACÉS PAR DES TYPES D'ENTITÉ.

Lors de la conception d'une base de données plusieurs situations peuvent amener le concepteur à transformer un type d'association en type d'entité. Les situations principales sont :

- Le type d'association est "many to many".
- Le type d'association est plus que binaire.
- Le type d'association contient un ou des attributs.
- Le type d'association n'est pas identifié par les rôles joués par les types d'entité sur lesquels il est défini.
- Le type d'association est cyclique.

Vu le nombre de ces situations et leur fréquence, il va de soi que l'on doive présenter ici les indices qui nous permettrons de détecter l'utilisation de cette tranformation.

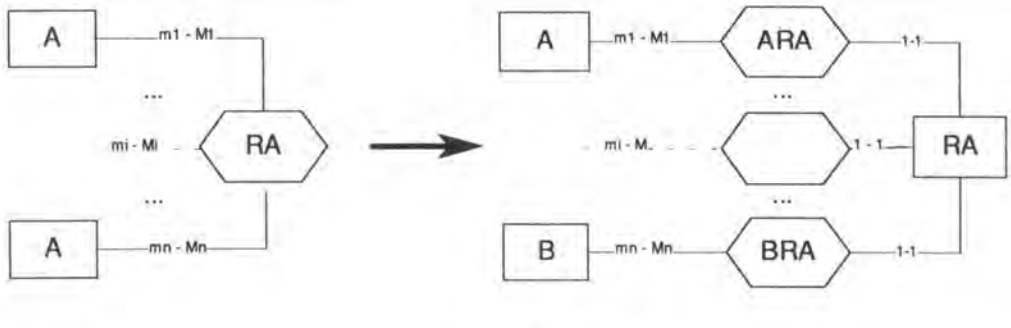


Figure V.1.

Comme indiqué à la figure V.1., un type d'entité né de la transformation d'un type d'association peut être facilement identifiable par les connectivités 1-1 de ses rôles. De plus nous pouvons observer les caractéristiques suivantes :

- Le type d'entité joue un rôle dans minimum deux types d'association. En réalité, le nombre de ceux-ci se limitera souvent à trois ou quatre.
- Le type d'entité ne possède pas ou peu d'attributs.
- Le type d'entité peut ne pas avoir d'identifiant, ou celui-ci est constitué d'attributs et d'un sous-ensemble des rôles.
- Le nom du type d'entité est obtenu par une technique identique à celle qui attribue un nom au type d'association.
- Il est possible aussi que le type d'entité soit en liaison avec un autre type d'entité via deux types d'association différents.

Description de la transformation dans le cadre de la rétro-ingénierie.

- Le type d'entité est remplacé par un type d'association.
- Ce type d'association est défini sur les types d'entité qui étaient associés au type d'entité transformé.
- La connectivité des rôles des types d'entité sur lesquels e nouveau type d'association est défini reste identique.

5.2. LA RECHERCHE DES RELATIONS ISA

5.2.1. LES RELATIONS ISA

Dans le modèle entité-association de base, on fait l'hypothèse que toute entité n'appartient qu'à un seul type d'entité. Or dans le mode réel, un objet peut être perçu comme appartenant à plusieurs types. Par exemple, comme nous le voyons sur la figure V.2., un train pourra avoir des caractèristiques générales, mais aussi des caractèristiques spécifiques aux trains de marchandises ou aux trains de voyageurs.

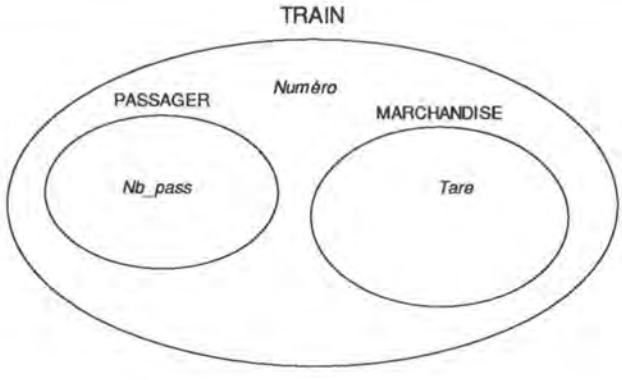


Figure V.2.

Donc, nous voyons qu'un train peut appartenir au type "TRAIN", ainsi qu'au sous-type "TRAIN-VOYAGEUR" ou "TRAIN-MARCHANDISE". On dira que le premier type est une généralisation des deux autres, qui eux en sont une spécialisation.

Pour modèliser de telles situations, nous aurons la relation d'inclusion ISA (IS-A). Dans la figure V.3., nous avons appliqué cette modèlisation à l'exemple de la figure V.2.

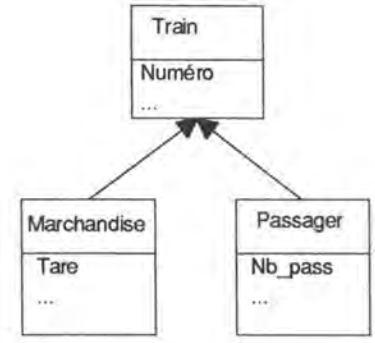


Figure V.3.

Nous pouvons parler de relation d'inclusion, car tout train de marchandises (ou de voyageurs) est également un train. Donc, les caractèristiques générales d'un train sont inclues dans les caractèristiques d'un train de marchandises (ou de voyageurs).

Notre démarche de rétro-ingénierie étant basée sur l'étude du comportement du concepteur de bases de données, nous ne pouvons pas nous permettre d'attribuer automatiquement les notions de couverture ou de disjonction aux sous-types détectés lors de nos recherches. Ces deux qualités ne seront ajoutées aux relations ISA que dans des cas précis que nous spécifierons lors de l'explication de nos méthodes de recherche.

5.2.2. La notion d'attribut virtuel

Avant de voir en détails les différentes méthodes de recherche, nous allons expliquer la notion d'attribut virtuel utilisé par certaines de celles-ci.

Un **attribut virtuel** est un attribut prenant la valeur VRAI ou FAUX et qui est ajouté momentanément (le temps d'une recherche) à un type d'entité. En réalité, on ajoutera un attribut virtuel pour chaque attribut facultatif. L'attribut virtuel prendra la valeur VRAI, pour une entité données, si l'attribut facultatif à une valeur autre qu'une représentation de la valeur nulle.

Exemple:

Si dans une entité, on ne prend en compte qu'un sous-ensemble des attributs virtuels, dans l'ordre de la déclaration des attributs facultatifs auxquels ils sont liés, on obtiendra l'expression d'un nombre binaire. La représentation de ce nombre binaire en base 10 est appelée la concaténation d'attributs viruels.

5.2.3. Recherche de relation ISA représentée au sein d'un type d'entité

Comme nous le voyons à la figure XII.4., une des manières d'implémenter une relation ISA, consiste à migrer les attributs des sous-types dans le type d'entité représentant le sur-type. De plus, nous remarquerons que les attributs ainsi ajoutés, sont facultatifs.

les exemples qui suivent sont dérivés de **[Bod89]**.

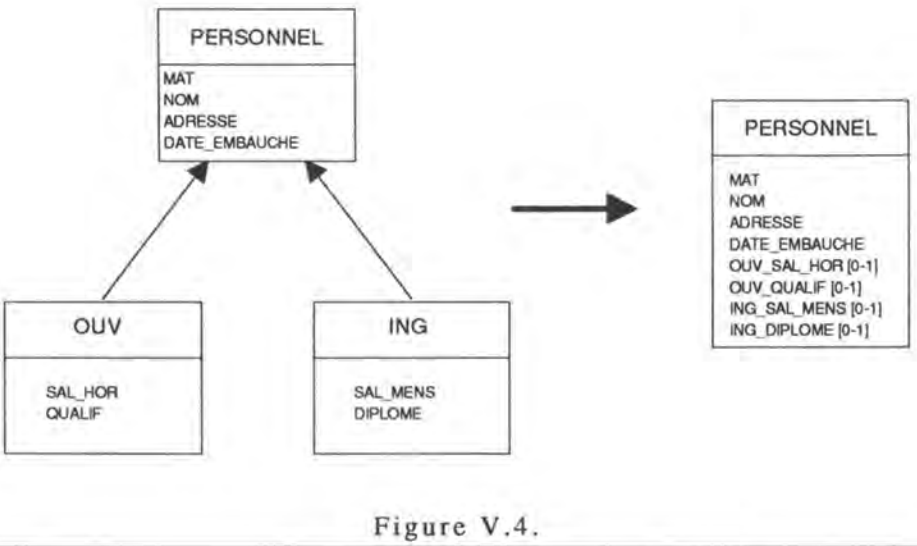


Figure V.4.

5.2.3.1. Recherche basée sur les sous-schémas

Une particularité des relations ISA est le fait que les sous-types hérite des attributs du sur-type. Donsc, si deux vues (sous-schémas) sont déclarées sur le type d'entité "PERSONNEL" de la figure XII.4.(droite), alors si chacune de ces vues reprend les attributs obligatoires du type d'entité plus une partie des attributs facultatifs et que ces parties sont disjointes, alors enfin on peut supposer les relations ISA de la figure XII.4.(gauche).

De manière plus formelle, nous rechercherons un type d'entité contenant plusieurs attributs facultatifs. Nous n'en garderons que ceux sur lesquels plusieurs vues sont définies. Si ces vues reprennent les attributs obligatoireset une partie des attributs faculatifs, alors on pourra supposer des relations ISA à condition que les ensembles d'attributs repris dans les vues soient disjoints.

Comme nous le remarquons, cette recherche ne permet pas au sur-type de posséder des attributs faculatifs. Pour combler cette lacune, nous pouvons ne pas tenir compte, dans la vérification de la disjonction, des attributs facultatifs mentionnés dans toutes les vues (ou sous-schémas).

Cette recherche possède trois limites; deux sous-types ne pevent posséder un même attribut, la qualité de disjonction est attribuée par défaut alors que la qualité de couverture n'est pas vérifiée.

De ce fait, il faudra soit se servir de cette recherche pour guider la recherche sur les données, soit faire intervenir l'opérateurpour un complément d'information et pour l'affirmation de l'existence des relations ISA.

5.2.3.2. RECHERCHE BASÉE SUR L'ÉTUDE DES NOMS.

5.2.3.2.1. DÉTECTION DE SOUS-TYPAGE PAR EXAMEN DES PRÉFIXES D'ATTRIBUTS

Cette méthode requiert la présence de plusieurs préfixes de noms d'attributs au sein du type d'entité.

Le cas où tous les attributs ont un préfixe commun, a été traité dans la partie concernant les préfixes.

5.2.3.2.1.1. PREMIER CAS

Tous les noms d'attributs ne sont pas préfixés.

Transformation par défaut :

- Les attributs non préfixés feront partie du type d'entité sur-type. Les attributs restants sont assignés, par groupes ayant même préfixe, au sein de types d'entité sous-types, dont le nom sera le préfixe commun à ces attributs (moins l'éventuel séparateur). Les préfixes sont retirés des attributs.

-Les relations entre les sous-types et le sur-type, s'excluent mutuellement.

L'utilisateur pourra à loisir modifier les noms de types d'entité, d'attributs, et la relation d'exclusion.

Ici le type d'entité PERSONNEL contient les préfixes suivants : OUV_, ING_

Le résulat sera :

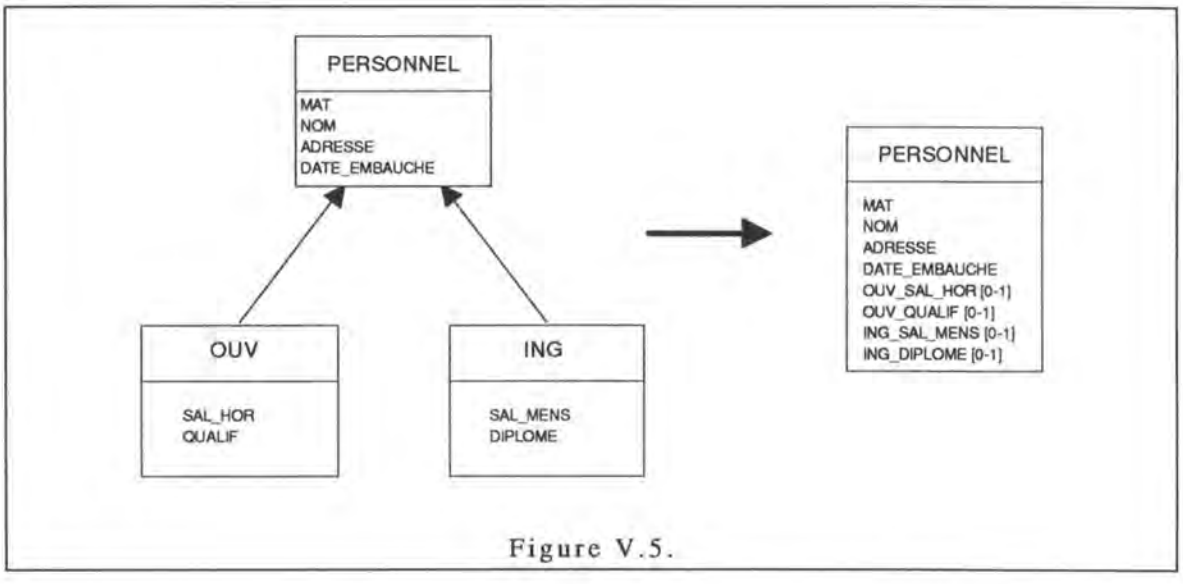


Figure V.5.

5.2.3.2.1.2. SECOND CAS

Tous les attributs sont préfixés, et les préfixes sont différents.

Transformation par défaut :

1) Mise en évidence du préfixe des attributs du sur-type, soit par :

- inclusion du préfixe dans le début du nom de type d'entité,

- similitude du préfixe et du nom de type d'entité,

- inclusion des lettres du préfixe dans le nom du type d'entité (cas où le préfixe est constitué des n premières consomnes du nom du type d'entité, par exemple),

- demande à l'utilisateur.

Les attributs partageant ce préfixe feront partie du type d'entité sur-type.

2) Les attributs restants sont assignés, par groupe ayant même préfixe, au sein de types d'entité sous-types, dont le nom sera le préfixe commun à ces attributs (moins l'éventuel séparateur). Les préfixes sont retirés des attributs.

- Les relations entre les sous-types et le sur-type, s'excluent mutuellement.

L'utilisateur, pourra à loisir modifier les noms de types d'entité, d'attributs, et la relation d'exclusion.

Résultat :

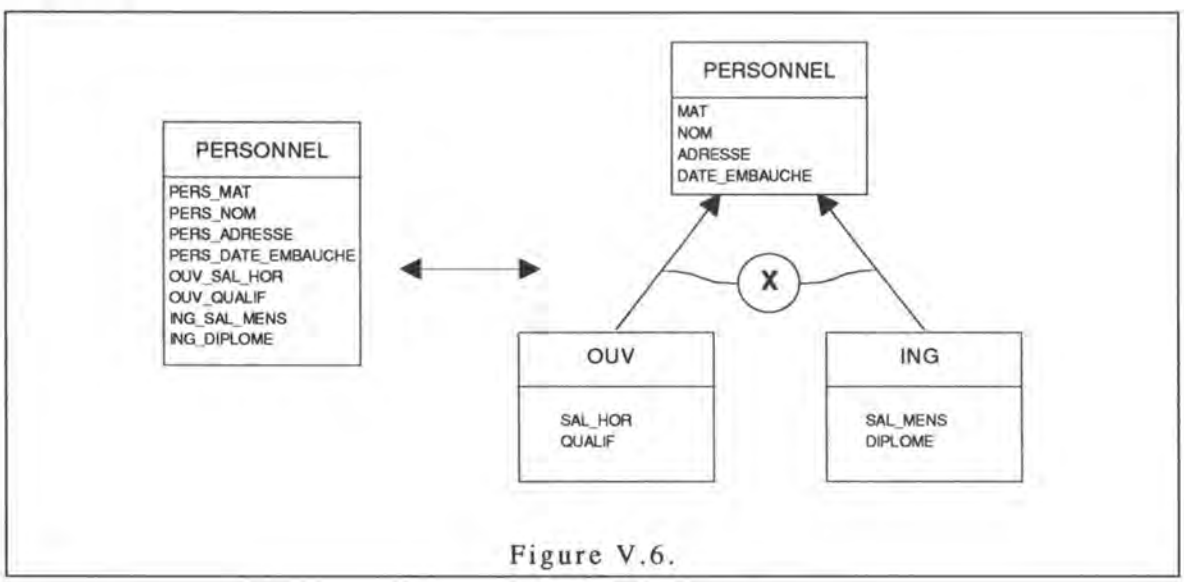


Figure V.6.

5.2.3.2.2. DÉTECTION DE SOUS-TYPAGE PAR EXAMEN DES SUFFIXES D'ATTRIBUTS

Le raisonnement est le même que pour la détection de sous-typage par examen des préfixes, c'est pourquoi nous ne nous y attarderons pas.

5.2.3.3. RECHERCHE BASÉE SUR L'ÉTUDE DES DONNÉES

5.2.3.3.1. SI IL Y A PRÉSENCE D'UN ATTRIBUT SPÉCIFIANT LE SOUS-TYPE

Cette détection de sous-typage est une démarche en 6 phases nécéssitant les concepts d'attributs virtuels et de concaténation d'attributs virtuels.

Première étape : Affectation des attributs du sur-type

Nous affecterons au sur-type, les attributs ne prenant jamais la valeur nulle.

Occurence	a	b	c	d	e	f
# 1	4	3	1	5	-	-
# 2	2	4	2	-	4	-
# 3	8	-	3	-	-	5
# 4	7	-	3	-	-	3
# 5	3	32	2	-	12	-
# 6	8	2	1	15	-	-

Tableau V.7.

Seconde étape : Recherche de l'attribut spécifiant le sous-type

Parmis les attributs du sur-type, nous rechercherons celui qui spécifie le sous-type. Pour ce faire, il faut mettre en évidence une dépendance fonctionnelle entre un attribut du sur-type et la concaténation d'un sous-ensemble d'attributs virtuels.

Occurence	a	c	b_d_e_f
# 1	4	1	12
# 2	2	2	10
# 3	8	3	1
# 4	7	3	1
# 5	3	2	10
# 6	8	1	12

Tableau V.8.

Troisième étape : Définition des sous-types

Le nombre de valeurs différentes de c signifie pour nous le nombre de sous-types. Les groupes d'attributs sont déterminés par la valeur de la concaténation. Par exemple, la valeur 12 représente les attributs b et d. Dans le cas où des valeurs différentes de l'attribut spécifiant le sous-type entraînent une même valeur de la concaténation, il sera demandé à l'utilisateur s'il désire regrouper les sous-types spécifiés par la même valeur.

Quatrième étape : Disjonction

Si l'ensemble des attributs d'un sous-type est l'union des ensembles d'attributs de plusieurs autres sous-types, alors il n'y aura pas de disjonction et le sous-type reprenant l'ensemble des attributs des autres pourra être éliminé.

Cinquième étape : Recouvrement

Concrètement, dans le cas où la concaténation prend la valeur 0, nous avons créés un sous-type ne contenant aucun attribut. En d'autres termes, cela représente le non recouvrement du sur-type par ses sous-types.

5.2.3.3.2. SI IL NY A PAS D'ATTRIBUT SPÉCIFIANT LE SOUS-TYPE

Les concepts d'attributs virtuels et de concaténation d'attributs virtuels sont également utilisés ici. La première étape de la démarche est identique à la première étape de la démarche précédente.

Première étape : Affectation des attributs du sur-type

Nous affecterons au sur-type, les attributs ne prenant jamais la valeur nulle.

Deuxième étape : Affectations des attributs aux sous-types.

Les attributs trouvés précédement feront partie du sur-type, nous n'en tiendrons plus compte. Il faut maintenant rechercher des ensembles d'attributs tels que :

$A_1,\ldots,A_n : \forall\ a_1,a_2 \in A_i\ (1 \leq i \leq n)$,

$\text{val}(\text{virt}(a_1))=\text{vrai} \Leftrightarrow \text{val}(\text{virt}(a_2))=\text{vrai}$

val(virt(a)) représente la valeur de l'attribut virtuel a

Occurence	c	d	e	f
# 1	1	vrai	faux	faux
# 2	2	faux	vrai	faux
# 3	3	faux	faux	vrai
# 4	3	faux	faux	vrai
# 5	2	faux	vrai	faux
# 6	1	vrai	faux	faux

Tableau V.17.

Le résultat sera ici 3 ensembles contenant respectivement les attributs d, e et f. ces 3 ensembles sont les types d'entité sous-types.

Troisème étape : Affectation des attributs restant

Il reste peut-être des attributs non classés, soit parce qu'ils contiennent des valeurs nulles et qu'il n'ont pu être inclus dans un ensemble, ou soit parce qu'ils sont communs à plusieurs sous-types. L'examen d'une de leurs tables de vérité permet d'inclure, si c'est possible, ces attributs, au sein d'ensembles.

Occurence	b	c	d	e	f
# 1	vrai	1	vrai	faux	faux
# 2	vrai	2	faux	vrai	faux
# 3	faux	3	faux	faux	vrai
# 4	faux	3	faux	faux	vrai
# 5	vrai	2	faux	vrai	faux
# 6	vrai	1	vrai	faux	faux

Tableau V.18.

Le résulat attribuera b aux ensembles contenant les attributs d et e, car :

b=vrai ⇔ d=vrai ou e=vrai.

5.2.4. Recherche de relations ISA représentées par plusieurs types d'entité

Nous venons de voir, au chapitre précédent, que pour représenter une relation ISA on pouvait faire migrer les attributs des sous-types dans le sur-type. Une autre solution est de faire migrer les attributs du sur-type dans les sous-types. Ceci donne la transformation suivante :

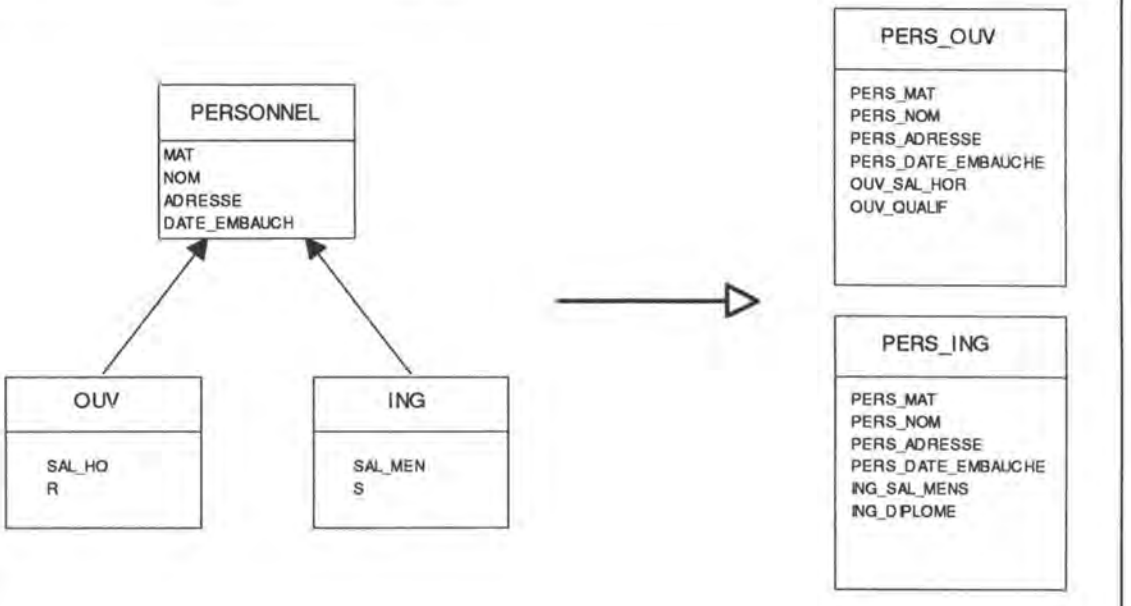


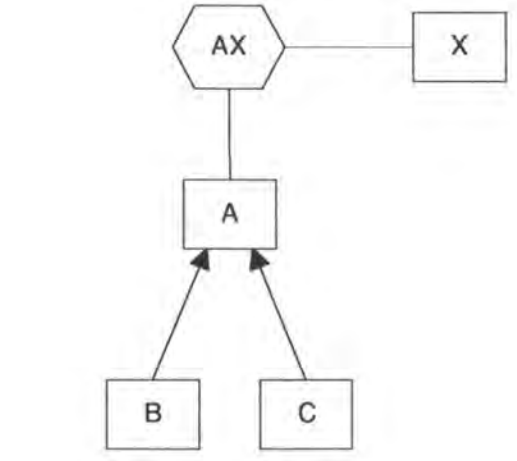
Figure V..

La principale source d'information pour détecter une telle transformation est l'analyse du schéma.

Premièrement, les types d'entité PERS_OUV et PERS_ING ont des noms possèdant un même préfixe. Ce préfixe commun provient de l'existence d'un sur-type commun.

Deuxièmement, un partie des attributs des deux types d'entité sont identiques (même nom, même domaine et mêmes caractéristiques). Si l'un d'eux (ou un ensemble de ceux-ci) est identifiant d'un type d'entité, il le sera aussi dans l'autre. De plus dans la plupart des cas ces attributs communs seront définis avant les attributs propres aux sous-types et dans le même ordre.

Enfin, on peut remarquer l'utilisation de cette transformation en analysant les types d'association. En effet, si le sur-type est lié par un type d'association à d'autres types d'entité, ces types d'association seront dédoublés et les rôles joués par ces autres types d'entité dans ceux-ci seront mutuellement exclusifs.



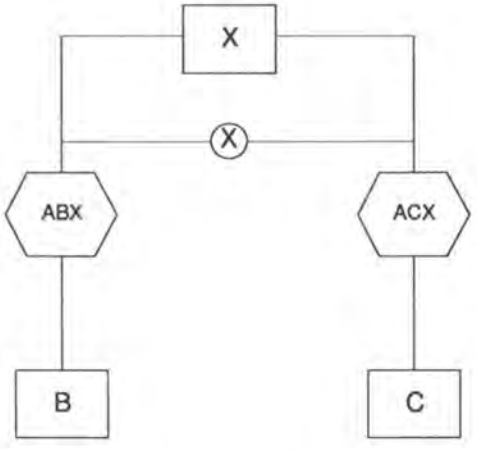


Figure V..

5.2.5. Recherche de relations ISA représentées par des types d'association

Après avoir vu comment représenter une relation ISA au moyen de types d'entité, nous présentons ici une transformation basée sur le concept de type d'association. Cette transformation est tirée de **[Bod89].**

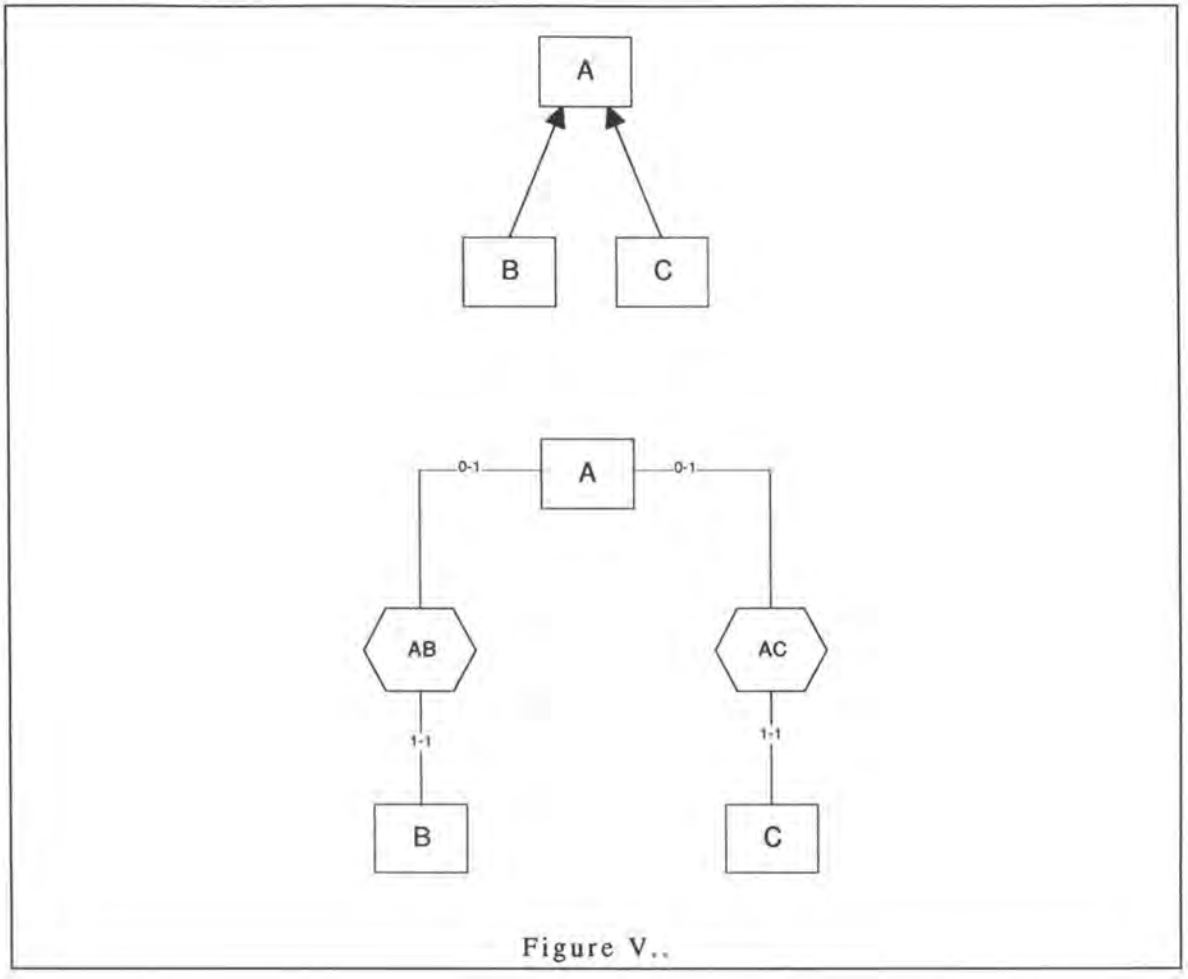


Figure V..

La forme du schéma de la partie inférieure de la figure ci-dessus paraît assez typique pour que l'on puisse se baser sur elle pour supposer une relation ISA. Mais un des éléments essentiels pour différencier une relation ISA d'un simple type d'association est la notion d'héritage.

Pour expliquer ceci, prenons l'exemple du type d'association "AB" de la dernière figure. Si e type d'association n'est pas l'implémentation d'une relation ISA, alors les modules d'accès au type d'entité "B" ne reprendront certainement pas les informations contenues dans "A". Par contre, si le type d'association "AB" est l'implémentation d'une relation ISA, alors tout module d'accès au type d'entité "B" devra également fournir les informations contenues dans le type d'entité "A".

6. UN OUTIL D'AIDE À LA CONCEPTION DES BASES DE DONNÉES: TRAMIS

Nous proposons au lecteur une brève description de l'atelier de conception de bases de donnés, TRAMIS, afin d'introduire la partie de notre mémoire visant à compléter ce logiciel, par quelques outils d'aide à la rétro-ingénierie.

Pour plus d'informations, nous renvoyons le lecteur à la consultation du travail de fin d'études de Monsieur Bruno RASE, [Ras92], et aux manuels d'utilisation de la première et seconde version de Tramis.

6.1. PRÉSENTATION DE TRAMIS

La conception de bases de données efficaces et opérationnelles est un processus complexe qui peut être assisté par un outil logiciel. La réalisation de la première version de TRAMIS (1991) avait pour but d'offrir au concepteur un ensemble d'outils destinés à l'appuyer dans cette tâche.

La seconde version de TRAMIS (1993), développée en c++, propose, en plus des outils de forward engineering, 4 transformations du domaine de la rétro-ingénierie .

6.1. PRÉSENTATION DE TRAMIS

La conception de bases de données efficaces et opérationnelles est un processus complexe qui peut être assisté par un outil logiciel. La réalisation de la première version de TRAMIS (1991) avait pour but d'offrir au concepteur un ensemble d'outils destinés à l'appuyer dans cette tâche.

La seconde version de TRAMIS (1993), développée en c++, propose, en plus des outils de forward engineering, 4 transformations du domaine de la rétro-ingénierie .

6.2. LE MODÈLE DE TRAMIS

L'espace de travail de tramis est divisé en systèmes. Chacun de ces systèmes regroupe 1 ou plusieurs schémas Entité-Association.

Les objets du schéma sont les types d'entité, les types d'association, les groupes et les collections d'entités.

6.2.1. LES TYPES D'ENTITÉ

Ils sont constitués d'un nom, d'un diminutif (shortname) et peuvent posséder des attributs multivalués et/ou facultatifs. Ces attributs peuvent également être décomposables ou non.

6.2.2. LES TYPES D'ASSOCIATION

Elles possèdent les mêmes caractèristiques que les types d'entité, en plus de deux rôles au moins. Ces rôles ont pour fonction de relier les types d'entité entre eux via les types d'association. Ils ont chacun un nom et une connectivité minimum et maximum.

6.2.3. LES GROUPES

Un groupe appartient à un type d'association ou à un type d'entité. Il est constitué d'attributs et/ou de rôles. Il sert à définir des clés simples, identifiantes ou de référence à un autre type d'association ou d'entité.

6.2.4. LES COLLECTIONS D'ENTITÉS

C'est le concept équivalent aux AREAs ou aux DBSPACEs des SGBD Codasyl et relationnels.

6.3. LE LANGAGE DE TRAMIS : ISL

Tramis possède son propre langage de description du modèle. Il décrit chacun des concepts de manière détaillée, facilitant ainsi la communication d'informations avec d'autres applications (les importateurs de schémas par exemple).

Pour plus de détails, nous invitons le lecteur à consulter **[Isl89]**.

6.4. LES POSSIBILITÉS DE TRAMIS 2

La version actuellement disponible de TRAMIS 2, n'est pas encore entièrement achevée, mais les fonctionnalités suivantes sont toutefois disponibles.

Outre les fonctions standards d'édition (undo, copy, paste, new et delete), TRAMIS propose 3 types de fonctions, faisant chacune l'objet d'un menu. La première concerne le schéma, la seconde la création d'objets et la dernière les transformations.

6.4.1. LE SCHÉMA

Ce menu offre la possibilité de créer, d'ouvrir, de fermer ou de copier un schéma. Il est d'ailleurs possible de modifier les propriétés d'un schéma.

6.4.2. LA CRÉATION

Les types d'objets manipulés dans tramis, et que l'on peut instancier ici sont : les types d'entité, les types d'association, les attributs, les rôles et les groupes.

La création d'un attribut au sein d'un type d'entité peut s'effectuer dans le niveau courant, ou en tant que composant d'un attribut décomposable.

Les groupes sont des listes d'attributs et/ou de rôles, qui auront fonction de cléf, de candidat identifiant, d'identifiant primaire,et seront soit de type direct ou chemin.

6.4.3. LES TRANSFORMATIONS

Les transformations s'effectuent à deux niveaux, soit sur un objet en particulier, ou sur l'entièreté du schéma.

6.4.3.1. SUR LES OBJETS

Un type d'entité peut être transformé en type de relation ou en attribut.

Un type de relation peut être transformé en type d'entité ou en attribut.

Un attribut peut être transformé en type de relation ou en type d'entité.

Un groupe peut être transformé en attribut (si tous ses composants sont des attributs), par aggrégation, ou en type de relation (si il fait l'objet d'une contrainte référentielle).

6.4.3.2. SUR LE SCHÉMA

Il est possible, grâce a cette fonction, de transformer :

- tous les attributs aggrégés en attributs simples,
- tous les attributs multi-valués en types d'entité,
- tous les types de relations complexes en types d'entité,
- tous les types de relations en attributs référentiels.

De plus, l'opportunité est offerte de transformer le schéma afin de le rendre compatible avec le modèle relationnel, et même, de générer sa description SQL.

6.5. MÉTA-SCHÉMA DE TRAMIS 2

Le méta-schéma de tramis, est le reflet du modèle utilisé.

Les traits plus épais terminés par une flèche représentent une relation *IS-A*, du sous-type vers le sur-type. Les traits en pointillés reliés à un petit cercle contenant un caractère représente un contraintes sur l'ensemble des objets ainsi désignés. Le "+" signifie l'exclusion. Enfin, les cadres en pointillés désignent des objets déjà définis sur une autre figure du méta-schéma.

Dans la figure VI.1. nous pouvons remarquer qu'un *system* est constitué de 0 ou N *schema*. Les *attributes*, les *entity-types* et les *relation-types* sont regroupés sous un même sur-type (*data_object*), partageant ainsi leurs caractèristiques communes. Ces *entity-types* et *relation-types* sont eux aussi regoupés au sein d'un type plus générral; *ent_rel_type*.

Les *collections* d'entités appartiennent au schéma et rassemblent 0 ou N *entity-types*.

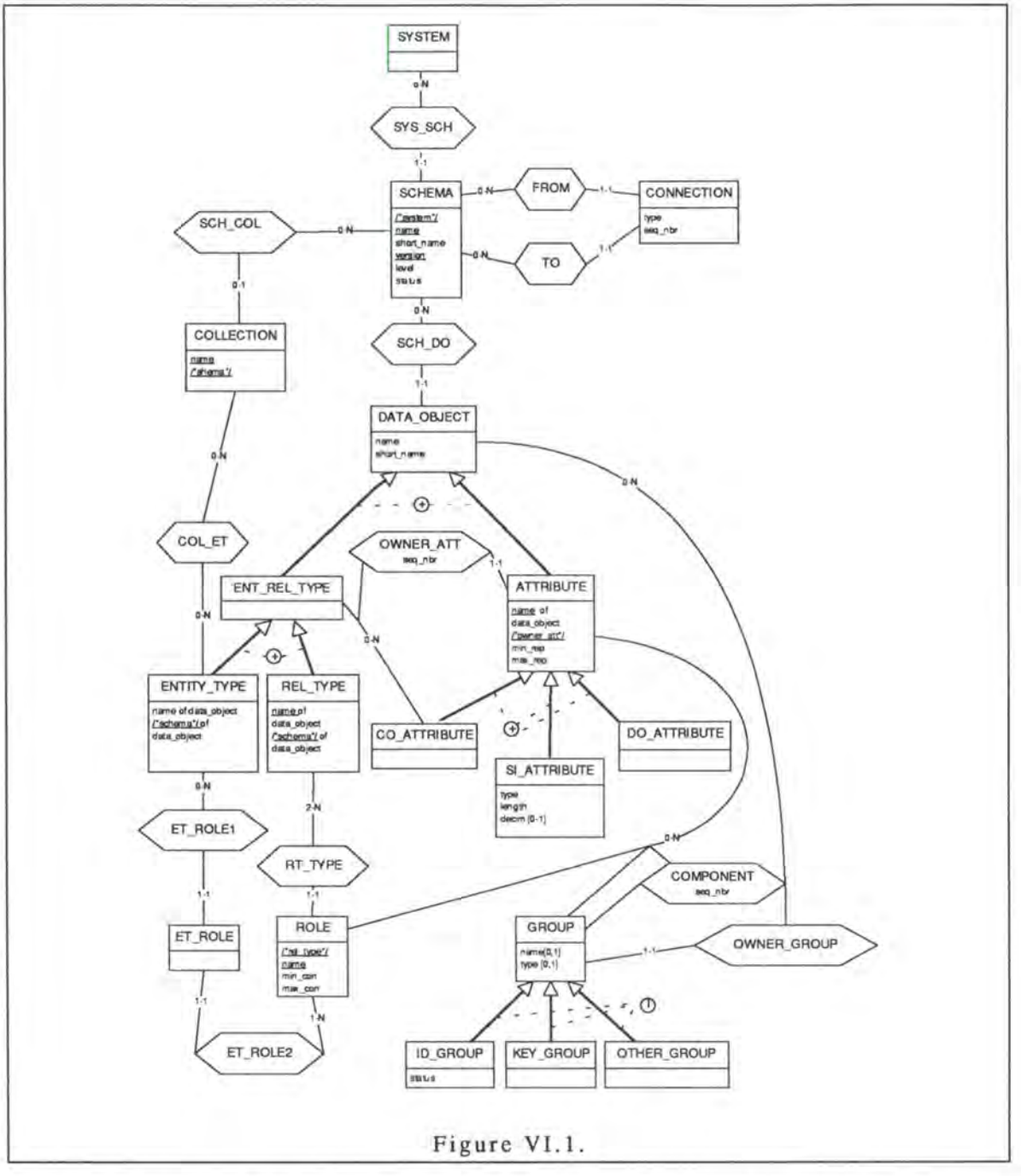


Figure VI.1.

La figure VI.2. représente la définition des domaines et leurs relations avec les objets du méta-schéma.

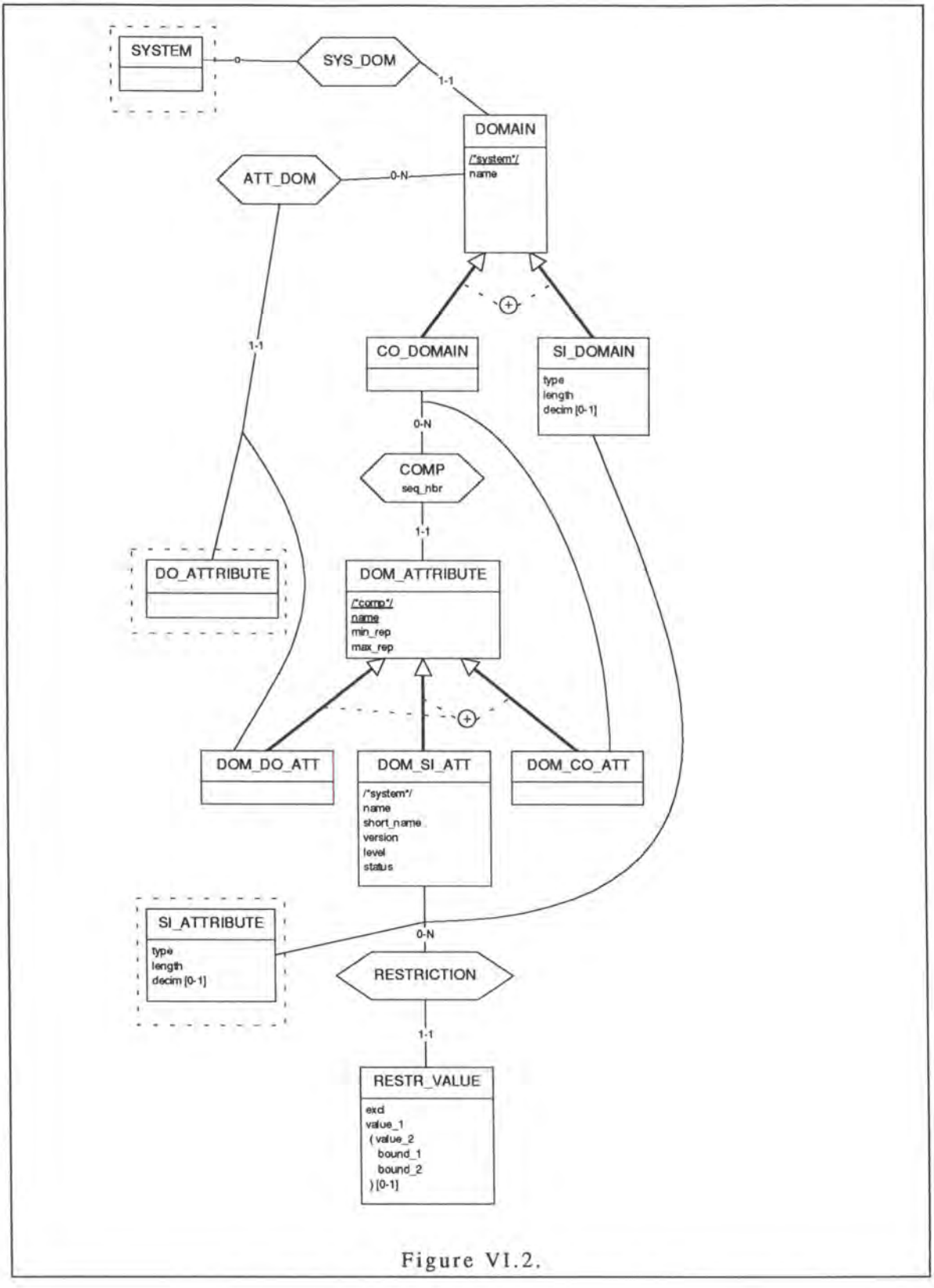

SYSTEM
SYS_DOM
1-1
DOMAIN
/*system*/
name
ATT_DOM
0-N
1-1
CO_DOMAIN
SI_DOMAIN
type
length
decim [0-1]
0-N
COMP
seq_nbr
1-1
DO_ATTRIBUTE
DOM_ATTRIBUTE
/*comp*/
name
min_rep
max_rep
DOM_DO_ATT
DOM_SI_ATT
/*system*/
name
short_name
version
level
status
DOM_CO_ATT
SI_ATTRIBUTE
type
length
decim [0-1]
0-N
RESTRICTION
1-1
RESTR_VALUE
excl
value_1
(value_2
bound_1
bound_2
) [0-1]

Figure VI.2.

La figure ci-après représente les différentes relations de sous-typages.

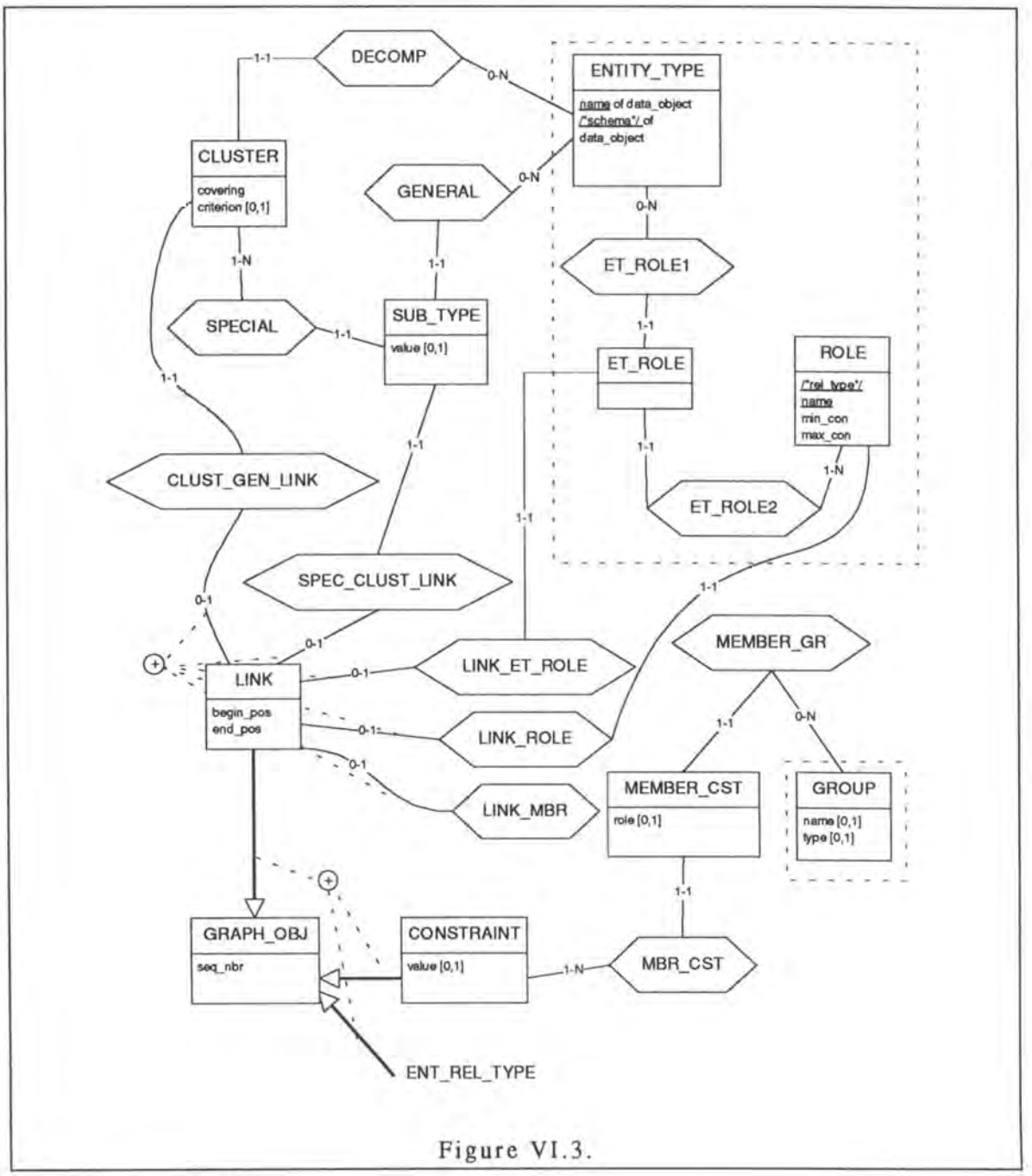


Figure VI.3.

6.6. NOTRE PARTICIPATION À TRAMIS 2

6.6.1. LES IMPORTATEURS.

Les modules d'importations ne font pas à proprement parler partie intégrante de l'atelier TRAMIS. Ils sont écrit en C standard, et ont comme but de transcrire la description d'une base de donnée CODASYL ou SQL en ISL (langage de description de schémas propre à TRAMIS).

6.6.1.1. L'IMPORTATEUR DE SCHÉMAS SQL.

Cet importateur se base sur la grammaire du SQL standard et plus particulièrement des SGBD SQL DB et ...

Les commandes comprises par l'importateur sont :

1. CREATE DATABASE + *def.*
2. CREATE SCHEMA + *def.*
3. CREATE TABLE + *def.*
4. CREATE FOREIGN KEY +*def.*
5. CREATE INDEX + *def.*
6. CREATE PRIMARY + *def.*
7. ALTER TABLE + *def.* (uniquement pour l'ajout d'une primary ou foreign key et l'ajout d'un index).

Le tableau ci-après montre les conversions effectuées sur les concepts SQL.

Concepts SQL	Concepts entités-associations	Commentaires
Database, schéma	schéma	
Table	*Entity-type*	
Colonnes	Attributs	Par défaut facultatif, obligatoire si NOT NULL.
Primary Key	Groupe identifiant	
Foreign Key	Groupe de référence	
Unique Index	Groupe clé d'accès	Si l'index est déclaré UNIQUE, et si il n'y a pas encore d'identifiant de même structure, on en contruit un.
Dbspace	Collection d'entités	

6.6.1.2. L'IMPORTATEUR DE SCHÉMAS CODASYL.

Le tableau ci-après montre les conversions effectuées sur les concepts CODASYL.

Concepts CODASYL	Concepts entités-associations	Commentaires
Schéma	Schéma	
Record-Type	*Entity-type*	
Data	Attributs	Par défaut obligatoires.Peuvent être décomposables et/ou multivalués
Set	Relation-Type	
Access Key	Groupe d'attributs	Suivant la nature de la définition, le groupe sera identifiant ou clé.
Area	Collection d'entités	

6.6.1.3. L'IMPORTATEUR DE SOUS-SCHÉMAS CODASYL.

La description d'un sous-schéma CODASYL est fort semblable à la définition d'un schéma CODASYL. Les principales différences sont :

1. Tous les concepts peuvent être renomés (un peu à la manière des *# define* du langage C).
2. La syntaxe des types est celle du langage COBOL.
3. Les sets ne sont pas décrits, ils sont juste énumérés.

Le *mapping* entre un sous schéma et un scéma global CODASYL se base sur les nom des *record-types* (types d'entité) et sur les noms d'attributs de premier niveau. La conversion des concepts est semblable à la conversion effectuée pour l'importation d'un schéma CODASYL.

6.6.2. TRANSFORMATION SEMI-AUTOMATIQUE DES TYPES D'ENTITÉ EN TYPES DE RELATION

Cette fonctionnalité est l'application d'une transformation déjà existante au sein de l'atelier, la transformation d'un type d'entité en type de relation.

La transformation s'adresse ici aux types d'entité succeptibles d'être, dans le schéma conceptuel d'origine, des types de relation. Elle consiste à trouver tous les types d'entité à même d'être transformés, c'est à dire répondant à un certain nombre de pré-conditions. Après confirmation de l'opérateur, la transformation est appliquée au type d'entité.

Pré-conditions :

- Tous les rôles joués par ce type d'entité sont 1-1.
- Aucun de ces rôles ne fait partie d'un *group*.
- Il existe un identifiant.

Post-condition :

- Le type d'entité devient un type de relation.
- Le type de relation conserve les attributs du type d'entité.

6.6.3. DÉ-PRÉFIXAGE SEMI-AUTOMATIQUE DES ATTRIBUTS DES TYPES D'ENTITÉ DU SCHÉMA

Lorsqu'un schéma est importé, il n'est pas rare de constater que les attributs sont tous préfixés par un diminutif du nom de leur type d'entité.

Le dé-préfixage des attributs concerne les attributs d'un type d'entité du schéma courant. Cette transformation s'applique aux attributs frères, c'est à dire de même niveaux. En d'autres mots, on recherche un préfixe pour les attributs du type d'entité et pour les composantes d'un attribut décomposable.

Pré-condition :

- Tous les fils d'un type d'entité ou d'un attribut décomposable ont le même péfixe

Post-condition :

- Le nom des attributs est le nom moins le préfixe (séparateur compris).

Remarque :

- Le dé-préfixage requiert toujours l'approbation de l'opérateur.

6.6.4. AFFINEMENT AUTOMATIQUE D'UN SCHÉMA À PARTIR DE SES SOUS-SCHÉMAS ET/OU VUES

La notion de sous-schéma variant d'un SGBD à l'autre, il nous à semblé juste de distinguer l'affinement d'un schéma global COSASYL à partir de ses sous schémas de l'affinement d'un schéma SQL à partir de ses vues.

Le processus d'affinement travail par défaut sur le schéma courant, qu'il considère comme le schéma global à affiner. Une fenêtre de dialogue demande à l'utilisateur le nom du sous-schéma à examiner ainsi que le nom du nouveau schéma global à construire et le type de SGBD d'où les schémas proviennent.

6.6.4.1. AFFINEMENT D'UN SCHÉMA CODASYL

Les analyseurs CODASYL se basent généralement sur les noms de records et d'attributs de premier niveau pour effectuer la correspondance entre le schéma global et les sous-schémas.

L'affinement du schéma global se déroule de la manière suivante:

1- On recopie le schéma global sous le nom du nouveau schéma global donné par l'utilisateur.

2- On se positionne sur le premier type d'entité du sous-schéma.

3- Dans le schéma global, on se place sur le type d'entité de même nom.

4- On se positionne sur le premier attribut du type d'entité du sous-schéma

5- Dans le schéma global on se place sur l'attribut de même nom.

6- Il y a mise à jour du type d'entité du schéma global si et seulement si une des deux conditions suivantes est remplie.

6.a.- *Si l'attribut du schéma global n'est pas multivalué mais l'attribut du sous-shéma oui.*

ou

6.b.- *En considèrant un attribut comme une arborescence, et le degré de définition de celui-ci comme la profondeur de l'arbre, si le degré de définition de l'attribut du sous-schéma est supérieur au degré de définition du schéma global.*

7- On passe à l'attribut suivant du type d'entité du sous-schéma. S'il n'y a plus d'attribut pour ce type d'entité on va au point 8, sinon au point 5.

8- On passe au type d'entité suivant du sous-schéma. S'il n'y a plus de type d'entité dans le sous-schéma, la recherche est finie, sinon on va au point 3.

6.6.4.2. AFFINEMENT D'UN SCHÉMA RELATIONNEL

Dans ce cas-ci, le sous-schéma rassemble toutes les vues portant sur le schéma global. On affine le schéma global vue par vue. La base de cette recherche est le texte SQL de la déclaration de vue qui figure en commentaire dans le type d'entité représentant cette vue dans TRAMIS. Ce que l'on recherche, est l'expression

d'attributs décomposables. A cette fin , on se base sur la syntaxe de deux cas de figure :

Une colonne d'une vue correspond plusieurs colonnes d'une table.

```
CREATE VIEW v_1 (..., entreprise , ...)
      AS SELECT  ....,
                 CONCAT ( organisme , régime )
                  ....
      FROM  entreprise
```

Une colonne d'une vue correspond une partie de colonne d'une table.

```
CREATE VIEW v_2 (..., jour ,mois , annee , ...)
      AS SELECT  ....,
                 SUBSTR ( date , 1 , 2 ),
                 SUBSTR ( date , 3 , 2 ),
                 SUBSTR ( date , 5 , 2 ),
                  ....
                 FROM assurance
```

Figure VI.4.

Le premier cas consiste à mettre en évidence plusieurs attributs d'une table de base qui représente un attribut décomposable. Le second, est la recherche d'un attribut d'une table de base qui peut être décomposé.

Nous remarquerons immédiatement que nous nous sommes limité aux vues exprimées sur une seule table de base. Ce choix se défend par le fait que les autres cas sont facilement réductibles à celui-ci.

7. CONCLUSION

Le moment est venu de conclure en analysant le travail effectué. A première vue, nous pouvons affirmer que l'objectif que nous nous étions fixé est atteint. De fait, toute personne désireuse de procéder à la rétro-ingénierie d'une base de données, qu'elle soit relationnelle ou de type CODASYL, pourra retrouver dans ces documents :

- Un ensemble de sources d'information qui pourraient être disponibles.
- La façon d'exprimer le schéma logique de la base de données qui l'occupe à partir de ces sources d'information. Sur ce point, nous remarquerons que nous n'avons pas pris comme hypothèse que l'on se trouvait dans le cas idéal où **toutes** les sources d'information sont disponibles. En effet, cette hypothèse nous a semblé trop restrictive et l'utilisation de notre travail en aurait été réduite à quelques cas purement académiques.
- Un ensemble de transformations utiles pour traduire le schéma logique en un schéma conceptuel. Pour ce faire, toutes ces transformations sont accompagnées d'heuristiques permettant de déceler la nécessité de leur utilisation. De plus, chaque heuristique a été critiquée au point de vue de son niveau de certitude.
- Un autre ensemble de transformations permettant d'exprimer le schéma conceptuel en termes de structures de données, d'un degré d'abstraction supérieur. Ces transformations sont également accompagnées d'heuristiques.

Le lecteur sera peut-être au regret de ne pas trouver dans ces pages une séquence précise de transformations à suivre pour procéder à la rétro-ingénierie d'une base de données. Nous nous défendons d'une telle critique par le fait qu'une séquence de cette sorte ne saurait être établie de manière définitive. En effet, il faudrait tenir compte du cas précis où l'on se trouve, et en particulier des sources d'information qui sont à notre disposition.

Enfin, le lecteur attentif aura peut-être remarqué une discordance entre la méthodologie décrite au second chapitre et l'application que nous en faisons. Effectivement, les théoriciens préféreront exploiter les sources d'information uniquement dans la phase d'extraction des structures de données pour garder l'aspect formel de la démarche. Les praticiens, eux, pour des raisons d'optimisation, exploiteront ces sources d'information aussi bien dans la phase d'extraction que dans la phase de conceptualisation des structures des données. En ce qui nous concerne, nous avons choisi d'extraire l'information des sources à l'endroit où elle est utile au raisonnement, et ce pour des raisons didactiques.

BIBLIOGRAPHIE

[Bat92] C. BATINI, S. CERI et S.B. NAVATHE, "Conceptual Database Design", Benjamin/Cummings.

[Bod89] F.BODART, Y.PIGNEUR, "Conception Assistée des Systèmes d'Information", Méthodes Informatiques et Pratiques (MIPS),Masson, seconde édition, 1989.

[Bul83] Cii Honeywell Bull, DPS 8, GCOS 8 Data Management-IV, Data Base Adminitrator, Reference Manual, Addendum A, 1983.

[Chi90] E. J. CHIKOFSKY, J. H. CROSS II, "Reverse Engineering and Design Recovery : A Taxonomy", IEEE Software january 1990, pages 13-17.

[Dat5] C.J. DATE, "Database Systems", volume I, cinquième édition, Addison/Wesley.

[Elm89] R. ELMASRI, S.B. NAVATHE, "Fundamentals Of Database Systems", Benjamin/Cummings.

[Fon92] M. M. FONKAM, W. A. GRAY, "An Approach to Eliciting the Semantic of Relational Databases", CAISE 92, pages 463-480.

[Hai1] M. JORIS, R. VAN HOE, J.-L. HAINAUT, E. CARDON, M. CHANDELON, C. TONNEAU, P. VERSCHEURE, F. BODART, F. VAN DAMME, M. VANWORMHOUDT, "Phenix : Methods and Tools for Database Reverse Engineering".

[Hai2] J.-L. HAINAUT, M. CADELLI, B. DECUYPER, O. MARCHAND, " Tramis : A Transformation Based Database CASE Tool".

[Hai86] J.-L. HAINAUT, "Conception Assistée des Applications Informatiques", Vol. 2: "Conception de la Base de Données", Masson, 1986.

[Hai89] J.-L. HAINAUT, "Introduction à la Théorie Relationnelle des Bases de Données", cours de seconde licence et Maîtrise en Informatique, Institut d'Informatique,F.N.D.P., Namur, 1989.

[Hai91] J.-L. HAINAUT, "Database Reverse Engineering : Models, technics and strategies", Institut d'Informatique, F.U.N.D.P., Namur, 1993.

[Hai92] J.-L. HAINAUT, M. CADELLI, B. DECUYPER, O. MARCHAND, "Database CASE Tool Architecture : Principles For Flexible Design Strategies"

[Hai93] J.-L. HAINAUT, "Schema Transformations For Database Engineering. Synthesis And Illustration", Institut d'Informatique, F.U.N.D.P., Namur, 1993.

[Hai93b] J.-L. HAINAUT, "Database Reverse Engineering : A Systematic Approach", Les nouvelles approches pour les bases de données, cours postgrades en informatique 1993, Institut d'Informatique, Facultés Universitaires de Namur.

[Isl89] "TRAMIS V1.0 - MANUEL DE RÉFÉRENCE", FUNDP & CONCIS, Novembre 1989.

[Perron] R. PERRON, "Design Guide For Codasyl Data Base Management Systems", Q.E.D.

[Phe] "Phenix Project Database Reverse Engineering Methodological Guide", version 1.

[Pre93] W. J. PREMERLANI, M. R. BLAHA, "An Approach for Reverse Engineering of Relational Databases", IEEE 03-93, pages 151-160.

[Qui92] R. QUIEVY, "Les Bases de Données Hétérogènes", Mémoire de Licence et Maîtrise, Institut d'Informatique, F.N.D.P., Namur, 1992.

[Ras92] B. RASE, Contribution à la Réalisation d'un Atelier de Conception de Base de Données, Mémoire de Licence et Maîtrise, Institut d'Informatique, F.N.D.P., Namur, 1992.

[Sql] IBM, "SQL/Data System", SQL Reference for IBM VM Systems and VSE, Version 3 Release 1.

[Syb] "Sybase : Reference Manual".